SYRACUSE UNIVERSITY

ELE 490 - INDEPENDENT STUDY:

ASPECTS OF A PHASED ARRAY RADAR FEED AND BIAS NETWORK DESIGN

Electrical Engineering

Date of Defense: 24th June 2013

Author: Gabriele Galiero Casay
Committee: Serhend Arvas

PROYECTO FIN DE CARRERA

TEMA: Aspectos de un phased array radar

TÍTULO: Diseño de las redes de alimentación y polarización

AUTOR: Gabriele Galiero Casay

TUTOR: Serhend Arvas

DEPARTAMENTO: L.C. Smith College

CENTRO DE LECTURA: Syracuse University

Fecha de Lectura: 24 Junio 2013

Calificación: A

RESUMEN DEL PROYECTO:

El desarrollo de nuevos tipos de radares emplea nuevas tecnologías basadas en fundamentos de radiación para mejorar su eficiencia y usabilidad. Dependiendo del tipo de finalidad o aplicaciones, para el cual ha sido diseñado el dispositivo radar, existen distintas soluciones. En primer lugar se realiza una introducción sobre la definición de radar, los tipos de radar que existen en base a diferentes parámetros y se concluye con una breve mención de las aplicaciones de cada uno en base a la frecuencia de la señal que emplean. Una de las características que diferencia un tipo de radar de otro es el tipo de antena que emplea para enviar al espacio la señal de radiofrecuencia. Este proyecto se centra en un tipo específico de radar, llamado phased array radar, en la banda S de frecuencia. La antena de un radar tiene una cierta directividad. Los radares en general hacen un barrido de 180 o de 360 grados, para localizar obstáculos y poder identificar el entorno que le rodea. No obstante, al ser fija la directividad de una antena es precisa la presencia de un sistema mecánico de rotación, para que la señal sea enviada en distintas direcciones. El principal inconveniente que presenta esta solución es la velocidad de barrido. En aplicaciones de seguimiento de objetos en movimiento o simplemente en casos en los que la identificación de los obstáculos debe efectuarse con mucha rapidez, el sistema de rotación no es lo suficientemente

rápido para cambiar la directividad de la antena de manera eficiente. Además, cabe mencionar que un sistema mecánico precisa un constante mantenimiento y hace que el volumen total del sistema de emisión de señal sea elevado, dificultando el transporte dela misma. Estas desventajas mencionadas anteriormente pueden ser subsanadas mediante el empleo de antenas de tipo phased array. Este tipo de antenas tienen la ventaja de poder cambiar su directividad empleando los fundamentos de array de antenas omitiendo así la presencia del sistema mecánico de rotación. La idea básica consiste en variar la fase de la señal de alimentación de cada elemento del array, que para este caso particular son dipolos microstrip de media onda. Dependiendo de la diferencia de fase que exista entre elementos consecutivos se obtiene una directividad o ángulo de lectura. Existen unos dispositivos empleados en radiofrequencia llamados desplazadores de fase. Estos dispositivos aplican a la señal de radiofrecuencia de entrada una determinada diferencia de fase en función de una señal continua de control. Con el objetivo de poder variar el ángulo de lectura se diseña un sistema electrónico de polarización de la antena. Dicho sistema debe ser capaz de generar las tensiones continuas de control, que se necesitan para obtener un determinado ángulo de lectura. La arquitectura del sistema de polarización consiste en generar mediante un programa de ordenador el ángulo de lectura. Para el diseño se emplea MATLAB, una potente herramienta que permite el procesado de matrices y vectores. A continuación mediante otras funciones definidas, de nuevo, en el entorno de MATLAB, se calculan las tensiones de control aplicables a cada dipolo. La antena empleada es un array de ocho dipolos, por lo que el sistema deberá proporcionar ocho tensiones continuas de control. Una manera sencilla de generar las tensiones de control necesarias es empleando un micro-controlador. Estableciendo una conexión entre el ordenador de origen y el micro-controlador es posible calcular las tensiones de control y enviarlas al micro-controlador, para que proporcione dichas tensiones a cada desplazador de fase de la antena. Existen muchas posibles soluciones para el planteamiento del sistema de polarización. No obstante se ha escogido como micro-controlador un Arduino Mega 2560. Este tipo de dispositivo es compatible con MATLAB, permite establecer una conexión con el puerto serie (USB) y posee 15 salidas analógicas para proporcional las señales de control. En cuanto al diseño del software empleado se van a desarrollar dos posibles soluciones. Por un lado editando de forma manual el fichero necesario para la conexión entre el ordenador y el micro-controlador. Por otro lado empleando un software predefinido disponible en la página oficial de MathWoks, llamado MATLAB Support Package for Arduino. Para finalizar el diseño del sistema de polarización se incluye una etapa amplificadora, no inversora, a cada puerto de salida del micro-controlador. Esto es debido a que el micro-controlador Arduino Mega 2560 puede proporcionar una tensión continua máxima de 5 voltios,

mientras que los desplazadores de fase necesitarán tensiones continuas que rondan los 10 voltios para proporcionar desplazamientos de fase de hasta 360 grados. En comparación con el sistema de mecánico de rotación, el sistema de polarización diseñado es capaz además de cambiar la directividad de la antena con mayor velocidad. Esto permitirá realizar barridos de 180 grados sin necesidad de mover la antena. Además dichos barridos se pueden realizar a altas velocidades, lo cual supone una ventaja para radares empleados para el seguimiento de objetos en movimiento. Con el objetivo de que dicho sistema sea entendible por cualquier usuario ajeno su diseño, se hace imprescindible la realización de una interfaz gráfica de usuario. Para ello se emplea la herramienta disponible en el entorno de MATLAB, GUIDE, que permite de forma muy intuitiva el desarrollo de interfaces gráficas para usuarios.

Como ya se mencionó anteriormente, se proponen dos soluciones al diseño de la red de polarización de la antena. Cada solución de diseño es sometida a una serie de pruebas mediante la plataforma de MATLAB. Se analizan las ventajas y desventajas de cada una de ellas.

Hasta el momento en el desarrollo del sistema de polarización de la antena no ha intervenido la señal de radiofrequencia, empleada para detectar los obstáculos. Dicha señal es generada por una fuente y debe proporcionarse en igual cantidad a cada uno de los ocho elementos del array. Con esta finalidad se diseña una red de alimentación en paralelo, que consiste en el empleo de divisores de potencia de radiofrequencia colocados en paralelo. Esto divide la señal proporcionada por la fuente en partes iguales, y las suministra a los elementos del array. Los divisores de potencia diseñados serán Wilkinsons de 3dB y 0º. Para su diseño se emplea la herramienta AWR2011, que realiza un análisis circuital para la obtención de resultados antes de su fabricación. Esto proporciona una solución aproximada. No obstante se precisaría además de un análisis electromagnético más completo para obtener los resultados de simulación muy próximos a los reales.

Para concluir con el estudio se obtienen una serie de conclusiones acerca de las ventajas que brindan los sistemas de alimentación y polarización diseñados trabajando como parte del sistema de un radar.

Abstract

Radar technologies have been developed to improve the efficiency when detecting targets. Radar is a system composed by several devices connected and working together. Depending on the type of radar, the improvements are focused on different functionalities of the radar. One of the most important devices composing a radar is the antenna, that sends the radio-frequency signal to the space in order to detect targets. This project is focused on a specific type of radar called phased array radar. This type of radar is characterized by its antenna, which consist on a linear array of radiating elements, in this particular case, eight dipoles working at the frequency band S. The main advantage introduced by the phased array antenna is that using the fundamentals of arrays, the directivity of the antenna can change by shifting the phase of the signal at the input of each radiating element. This can be done using phase shifters. Phase shifter consists on a device which produces a phase shift in the radio-frequency input signal depending on a control DC voltage. Using a phased array antenna allows changing the directivity of the antenna without a mechanical rotating system.

The objective of this project is to design the feed network and the bias network of the phased antenna. The feed network consists on a parallel-fed network composed by power dividers that sends the radio-frequency signal from the source to each radiating element of the antenna. The bias network consists on a system that generates the control DC voltages supplied to the phase shifters in order to change the directivity. The architecture of the bias network is composed by a software, implemented in Matlab and run in a laptop which is connected to a micro-controller by a serial communication port. The software calculates the control DC voltages needed to obtain a determined directivity or scan angle. These values are sent by the serial communication port to the micro-controller as data. Then the micro-controller generates the desired control DC voltages and supplies them to the phase shifters. In this project two solutions for bias network are designed. Each one is tested and final conclusions are obtained to determine the advantages and disadvantages. Finally a graphic user interface is developed in order to make the system easy to use.

Resumen

Las tecnologías empleadas por lo dispositivos radar se han ido desarrollando para mejorar su eficiencia y usabilidad. Un radar es un sistema formado por varios subsistemas conectados entre sí. Por lo que dependiendo del tipo de radar las mejoras se centran en los subsistemas correspondientes. Uno de los elementos más importantes de un radar es la antena. Esta se emplea para enviar la señal de radiofrecuencia al espacio y así poder detectar los posibles obstáculos del entorno. Este proyecto se centra en un tipo específico de radar llamado phased array radar. Este tipo de radar se caracteriza por la antena que es un array de antenas, en concreto para este proyecto se trata de un array lineal de ocho dipolos en la banda de frequencia S. El uso de una antena de tipo phased array supone una ventaja importante. Empleando los fundamentos de radiación aplicado a array de antenas se obtiene que la directividad de la antena puede ser modificada. Esto se consigue aplicando distintos desfasajes a la señal de radiofrecuencia que alimenta a cada elemento del array. Para aplicar los desfasajes se emplea un desplazador de fase, este dispositivo aplica una diferencia de fase a su salida con respecto a la señal de entrada dependiendo de una tensión continua de control. Por tanto el empleo de una antena de tipo phased array supone una gran ventaja puesto que no se necesita un sistema de rotación para cambiar la directividad de la antena.

El objetivo principal del proyecto consiste en el diseño de la red de alimentación y la red de polarización de la antena de tipo phased array. La red de alimentación consiste en un circuito pasivo que permite alimentar a cada elemento del array con la misma cantidad de señal. Dicha red estará formada por divisores de potencia pasivos y su configuración será en paralelo. Por otro lado la red de polarización consiste en el diseño de un sistema automático que permite cambiar la directividad de la antena. Este sistema consiste en un programa en Matlab que es ejecutado en un ordenador conectado a un micro-controlador mediante una comunicación serie. El funcionamiento se basa en calcular las tensiones continuas de control, que necesitan los desplazadores de fase, mediante un programa en Matlab y enviarlos como datos al micro-controlador. Dicho micro-controlador genera las tensiones de control deseadas y las proporciona a cada desplazador de fase, obteniendo así la directividad deseada. Debido al amplio abanico de posibilidades, se obtienen dos soluciones que son sometidas a pruebas. Se obtienen las ventajas y desventajas de cada una. Finalmente se implementa una interfaz gráfica de usuario con el objetivo de hacer dicho sistema manejable y entendible para cualquier usuario.

ACKNOWLEDGMENTS

First and foremost I would like to thank my advisor Mr. Serhend Arvas for his guidance and support that has helped to successfully complete my bachelor's degree thesis. In addition I would like to thank Mr. Rafael Herradón, subdirector of international relationships of Universidad Politécnica of Madrid, for his continuous advice and allowing me the opportunity of studying abroad to finish my studies in Syracuse University.

Finally I would like to thank my family for their motivation and encouragement during the year. I thank all my colleagues from Spain and from Syracuse for their support and for making my studies an enjoyable experience.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In this chapter a general definition of radar will defined. It will also be shown how it is composed as well as its most important design parameters. Based on the properties of the radar some types of radar will be discussed and finally the most general applications of each type or radar will be mentioned.

1.1. Definition of RADAR

The word radar was originally an acronym, RADAR, for Radio Detection and Ranging. Range in radar communications denotes the distance from the radar to the target.
In general, radars are sensors that send electromagnetic waves into a specific region of space for the detection and location of reflecting objects. The radar sends the electromagnetic waves to the space making use of antennas or antenna systems. When these electromagnetic waves intercept some object (target) within the region they are sent, a portion of the power is reflected and comes back to the radar. These portions of electromagnetic energy are called echoes or radar returns. The receiver has to decide after, amplification of the received signal, if there is or not a target echo.
Once the echoes are identified at the output of the receiver, they are processed to extract target information such as range, velocity, dimension, and other characteristics. In the following figure a scheme of a simple radar is shown.

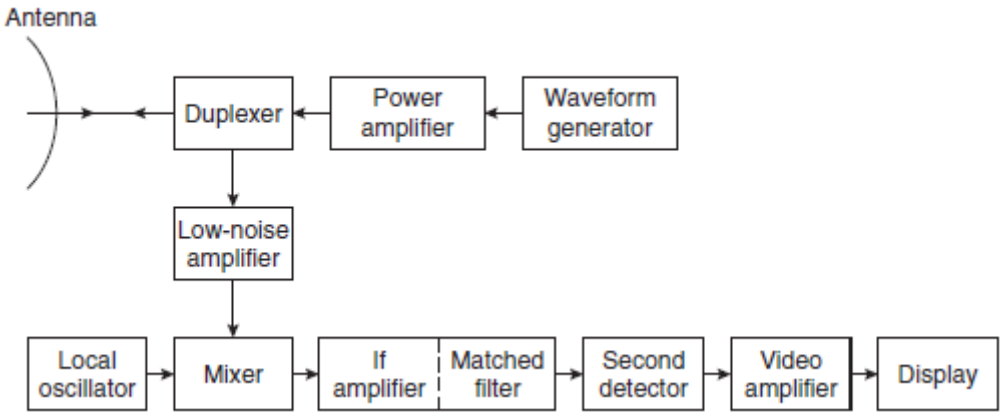


Figure 1.1. Scheme of a simple radar using superheterodyne receiver.

Each part of the system has different purposes:

- Transmitter: It is composed by a wave generator and a power amplifier. The transmitter has several functions:

- Generates the signal with the corresponding waveform, depending on the type of radar.

- Modulates the base-band signal, that is going to be used to detect targets, with an RF signal. The base-band signal is modulated because of the propagation characteristics of the waves in the air are better than others, such as waveguides or transmission lines.

- Sets the average power to be transmitted depending on the sensibility of the radar, which means the minimum power that the receiver is able to detect, due to the radar noise. The received power signal must be greater than the sensibility, otherwise the receiver will not detect the echoes.

- Duplexer: Allows bi-directional communication over a single path. It isolates the receiver from the transmitter while permitting them to share a common antenna. It avoids the transmitted signal goes to the detector and breaks it down.

- Antenna: Interface device that allows the electromagnetic waves propagates over the space. By the reciprocity theorem, the behavior of the antenna is the same to receive signal echoes. Then using the parameters of the antenna such as directivity, the transmitter can concentrate the electromagnetic power sent into a specific direction. Also it is possible to change the directivity of the antenna automatically to obtain the target information from several directions.

- RF LNA: A Low-Noise Amplifier is used because of the power of the incoming signal echoes are usually small due to the attenuation of the space, the insertion losses in the objects, diffraction and so on. It has to be low noise level in order to avoid the corruption of the echoes that have low level of power.

- Receiver: It is composed by a detector of the incoming echoes and digital system to process the information of the signal received. The receiver has to decide if the incoming signals are valid echoes or not (clutter) at the output, prior to process the signal to obtain the information. In order to make the decision, the output of the receiver has to be greater than a threshold previously established.

    - Detector: It is the most important part of the receiver. In the above figure the detector is a superheterodyne receiver. A superheterodyne receiver demodulates the incoming signal into an intermediate frequency and amplifies it. After that, the obtained signal is demodulated to obtain the band-base signal, that contains the information it is going to be processed.

    - IF amplifier: Amplifies the intermediate demodulated echoes.

- Matched Filter: Is used to improve the signal-noise ratio.

- Display: Eventually, when the receiver determines that the received echo is valid, it proceeds to extract the information in order to display it.

1.2. Radar parameters

Radars have several parameters to detect targets around it and display them in a screen.

- Range: Range is the distance from the radar to the target. The radar calculates the range, R, from the time delay, Δt of the echoes. The time delay is the time that the signal takes to do a round trip.

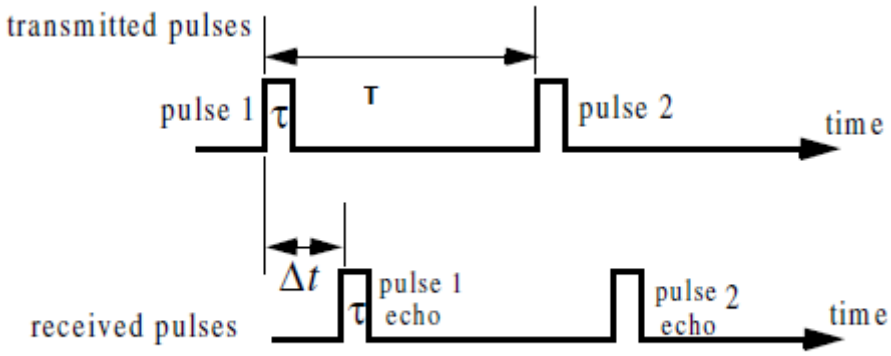


Figure 1.2. Time delay between the transmitted pulse and the received pulse.

The range is easily calculated by the following equation:

$$R = \frac{c\Delta t}{2} \tag{1.1}$$

The factor $1/2$ is present because of the signal travels two times the range distance before arriving to the radar.

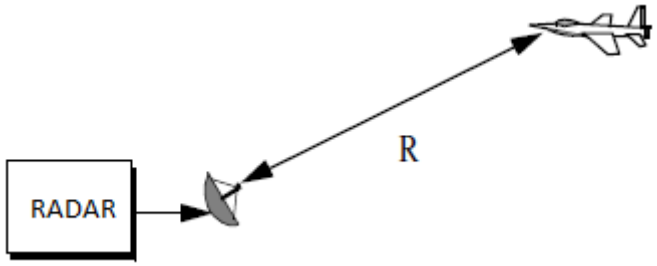


Figure 1.3. Scheme of a radar detecting targets.

- Range Resolution: It describes the ability of the radar to distinguish between two or more different targets on the same bearing but at different ranges. The range resolution depends mainly on the width of the pulse, τ, but also depends on the size and distance between targets. A well-designed radar should be able to distinguish targets separated one-half of the pulse width. Then the expression of the range resolution can be written as follows:

$$S_r = \frac{c\tau}{2} \tag{1.2}$$

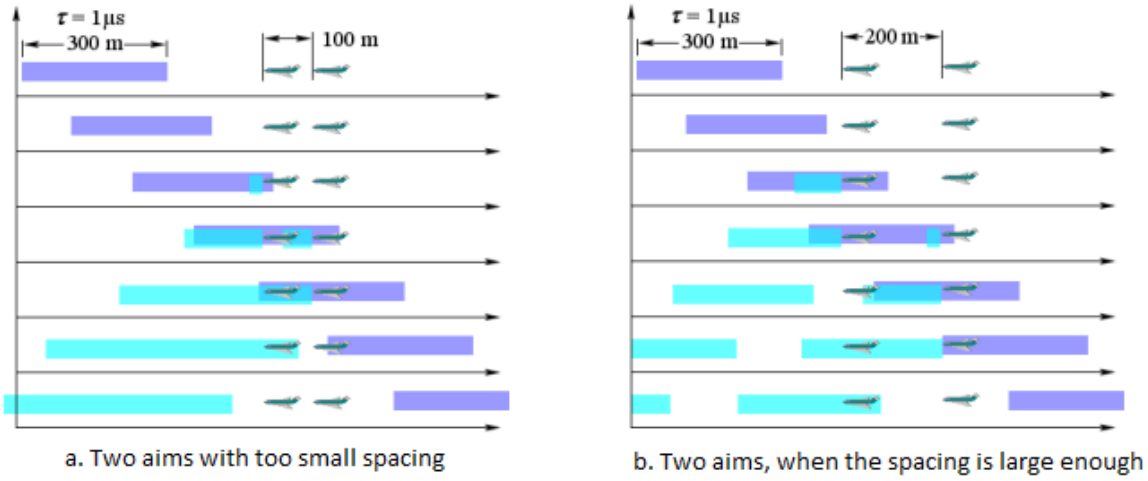


Figure 1.4. Scheme of two different scenarios in detection of multiple targets.

In the figure above it can be noted that if the two targets are so close, it can be superposition of echoes (Fig. 1.a). But if the targets are far enough there is no superposition between echoes (Fig. 1.b).

- Doppler Frequency: The Doppler phenomenon consists of a shift in the incident waveform due to the target motion with respect to the source of radiation. The Doppler frequency shift can be written as:

$$f_d = \frac{2v\cos\theta}{\lambda} \tag{1.3}$$

Where λ is the radar wavelength, θ is the angle between the direction of target and the radar beam.

1.3. Types of Radar

Radars can be classified into several categories depending on its characteristics, such as frequency band, antenna type and waveform used.

1.3.1. Frequency Band Radars

The Institute of Electrical and Electronic Engineers (IEEE), divided the frequency spectrum into several frequency bands. Based on it the International Communications Union (ITU) established specific ranges of frequency within each frequency band which can be used to radar applications.

Frequency Band	Nominal Frequency Range	Frequency Ranges for radar applications based on ITU
VHF	30-300 MHz	138-144 MHz 216-225 MHz
UHF	300-1000 MHz	420-450 MHz 890-942 MHz
L	1.0-2.0 GHz	1.215-1.4 GHz
S	2.0-4.0 GHz	2.3-2.5 GHz 2.7-3.7 GHz
C	4.0-8.0 GHz	4.2-4.4 GHz 5.25-5.925 GHz
X	8.0-12.0 GHz	8.5-10.68 GHz
Ku	12.0-18.0 GHz	13.4-14.0 GHz 15.7-17.7 GHz
K	18.0-27.0 GHz	24.05-24.25 GHz 24.65-24.75 GHz

Table 1.1. Frequency band assignations for radar applications based on ITU.

1.3.2. Waveform Radars

Radars can be also classified by the waveform used by the transmitter. Based on this classification we can distinguish several types of radars. In this section only some are mentioned.

- Pulse radar: This is the simplest radar. It radiates approximately rectangular pulses.

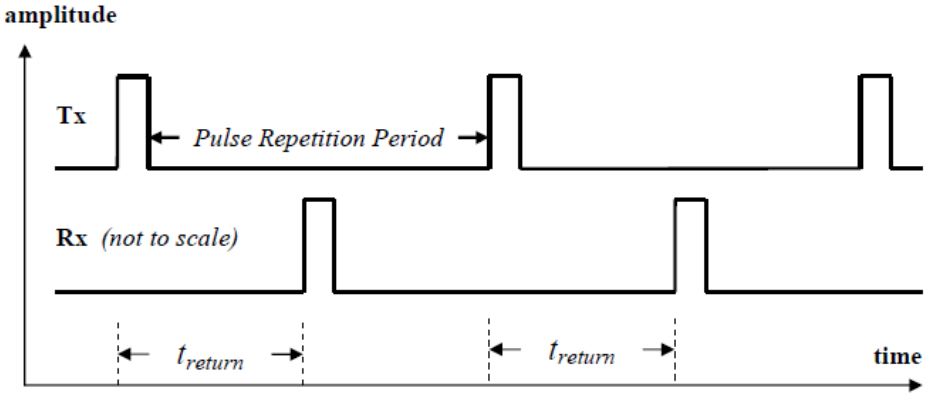


Figure 1.5. Detection signal used in pulse radar.

- Pulse compression radar: This type of radar radiates long modulated pulses using frequency or phase modulation. The purpose of using this modulation is to obtain the energy of a long pulse with the resolution of a short pulse.

- Continuous Wave (CW) radar: A sine wave shaped is radiated by the radar. In addition the doppler frequency shift is often used to detect targets in motion.

- Frequency Modulation Continuous Wave (FM-CW): A frequency modulation is used to allow an improved range measurement.

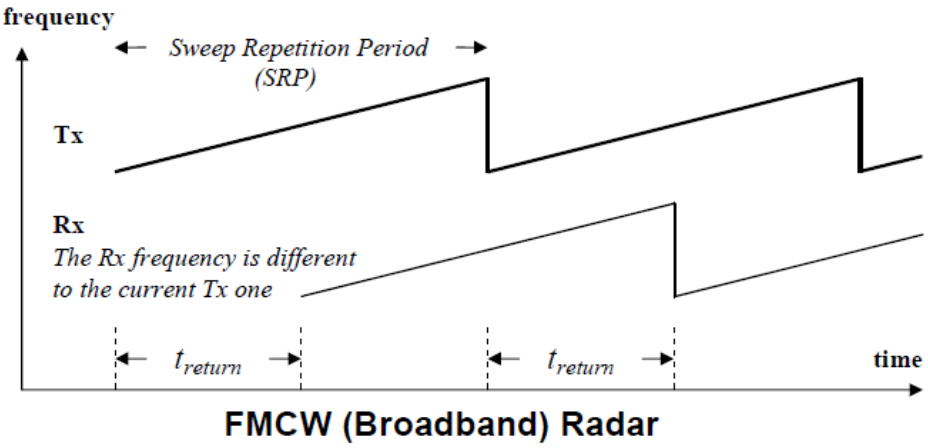


Figure 1.6. Detection signal used in FM-CW radars.

1.3.3. Tracking Radars

There are several types of radar based on whether or not are able to track or provide the trajectory of targets.

- Single Target Tracking (STT): Tracks a simple target with enough resolution providing an accurate tracking.

- Automatic Detection and Tracking (ADT): It is able to track several targets. This feature is reached by the measurements of the target locations obtained by several scans of the antenna.

- Phased array tracker: Tracks more than one targets with an electronically scanned phased array. The concept of phased array, that is the topic of this study, it is going to be developed later.

1.4. Radar applications

Depending on the frequency band, there are many radar applications:

- VHF: The first radars were developed to work at this range of frequencies. They were used to military purposes such as detection of ballistic missiles or long range air surveillance. Long range is reached because of the reflection coefficient from earth's surface is very large at this frequencies, so that the constructive interference of direct signal and the reflected signal from the earth increases the range of detection.
  The main inconvenience of this type of radar is the interference with signals generated by other communication systems operating at the same frequencies such as FM and TV broadcast.

- UHF: The main application of this type of radar is long range detection of missiles too. Besides are used to detect and track satellites.

- L band: This band is the most used to long-range air surveillance radars, but the rain could cause problems. That is due to the attenuation of waves at this frequency in presence of the rain. In addition it can be used to detect satellites and ballistic missiles.

- S band: It is used in air surveillance systems for long-range detection, because of measurements are more accurate at this frequency band, although the range is less than VHF and UHF radars. In addition this type of radar is more suitable to obtain a 3D vision of targets. One more application is for near and far weather observation.

- C band: It has similar characteristics and applications as that of S and X band radars.

- X band: This radar band is mostly used in military applications, such as airborne radars for performing an interceptor, fighter and attack. It is also used to civil marine radars, missile guidance and airborne Doppler navigation radars.

- K,K_u band: This frequency band has high range of frequencies, this makes the size of the antennas smaller than X band radar antennas. These radars are used in military airborne as the X band radar. The range of the K band at 22 GHz there is a water absorption that causes an additional attenuation that can limit the capability of range detection.

2. RADAR EQUATION

Before defining the radar equation, it is important to recall some definitions of radiation theory. Radiation intensity is the power radiated in a given direction per unit solid angle and can be written as follows:

$$U(\theta,\phi) = \frac{1}{2}\mathbb{R}e(E \times H^*) \cdot r^2\hat{r} = \frac{|E|^2}{2\eta}r^2 \tag{2.1}$$

Where the wave is a spherical wave and the expressions of fields are:

$$\vec{E}(r,\theta,\phi) = \vec{E}(\theta,\phi)\frac{e^{ikt}}{r} \tag{2.2}$$

$$\vec{H}(r,\theta,\phi) = \vec{H}(\theta,\phi)\frac{e^{ikt}}{r} \tag{2.3}$$

The relationship between them is shown in the following equation:

$$\vec{H}(\theta,\phi) = \frac{1}{\eta}\hat{r} \times \vec{E}(\theta,\phi) \tag{2.4}$$

Where η is the intrinsic impedance of the medium.

The total power radiated can be expressed as:

$$P = \iint \frac{1}{2}\mathbb{R}e(E \times H^*) \cdot \overrightarrow{dS} = \iint \frac{|E(\theta,\phi)|^2}{2\eta r^2}r^2 sin\theta d\theta d\phi = \iint \frac{|E(\theta,\phi)|^2}{2\eta} sin\theta d\theta d\phi$$

That is equal to the next equation:

$$P = \iint U(\theta,\phi)d\Omega \tag{2.5}$$

Considering a radar with an isotropic antenna, the power density in the direction of the maximum gain is given by:

$$U(\theta,\phi) = U_{iso} \tag{2.6}$$

That means the radiation density is constant, the power density is independent on the direction. Therefore the total power radiated by an isotropic antenna is:

$$P_r = \iint U(\theta,\phi)d\Omega = U_{iso}\iint d\Omega = U_{iso}\int_{\phi=0}^{2\pi}\int_{\theta=0}^{\pi} d\theta d\phi = U_{iso}4\pi \tag{2.7}$$

The total power radiated by an isotropic antenna is expressed as follows:

$$U_{iso} = \frac{P_r}{4\pi} \tag{2.8}$$

At this point the expression of the power density at a range R can be derived:

$$\left|\vec{S}(R,\theta,\phi)\right| = \left.\frac{U(\theta,\phi)}{r^2}\right|_{r=R} = \frac{P_r}{4\pi R^2} \tag{2.9}$$

It is known that there are no isotropic antennas in practice. For that reason the gain has to be included in the equation. The directivity of the antenna is defined by:

$$D(\theta,\phi) = \frac{U(\theta,\phi)}{U_{iso}} = \frac{U(\theta,\phi)}{P_r}4\pi \tag{2.10}$$

The same relationship for a loss-less antenna can be expressed as:

$$U(\theta,\phi) = U_{iso}D(\theta,\phi) \tag{2.11}$$

The gain of the antenna is defined by:

$$G(\theta,\phi) = \frac{U(\theta,\phi)}{P_{in}}4\pi \tag{2.12}$$

Where P_{in} is the power at the antenna input. Generally it is smaller than the radiated power due to the conductor loss. Both the input and radiated power are related by the efficiency:

$$e = \frac{P_r}{P_{in}} \leq 1 \tag{2.13}$$

If the antenna has no losses, $e = 1$.

Considering an antenna with losses the radiation intensity can be expressed in function of the gain and the radiation intensity of an isotropic antenna:

$$U(\theta,\phi) = U_{iso}G(\theta,\phi) \tag{2.14}$$

The power density at a range R, with a non-isotropic antenna is:

$$P_D = \frac{U(\theta,\phi)}{r^2}\bigg|_{r=R} = \frac{U_{iso}G(\theta,\phi)}{R^2} = \frac{P_r}{4\pi R^2}G(\theta,\phi) \tag{2.15}$$

When the electromagnetic wave radiated by the radar intercepts with a target, surface currents are induced on it and radiates electromagnetic waves in all directions. The amount of the power radiated is proportional to the target size, material, orientation and shape. All these parameters are integrated in a target parameter called Radar Cross Section (RCS) and is denoted by σ.
The radar cross section is defined as the ratio of the power reflected back to the radar to the power density incident on the target:

$$\sigma = \frac{P_R}{P_D}\ (m^2) \tag{2.16}$$

Where P_R is the power reflected by the target. Therefore the power intercepted by the target is:

$$\frac{P_t}{4\pi R^2}G(\theta,\phi)\sigma\ (watts) \tag{2.17}$$

The power travels again a distance R to arrive to the radar and the received power density becomes:

$$\frac{P_t}{4\pi R^2}\frac{1}{4\pi R^2}G(\theta,\phi)\sigma = \frac{P_t}{(4\pi R^2)^2}G(\theta,\phi)\sigma\ (watts/m^2) \tag{2.18}$$

The gain is considered in the direction of maximum radiation:

$$G = G_{max}(\theta,\phi) \tag{2.19}$$

The equation (2.18) becomes:

$$\frac{P_t}{(4\pi R^2)^2}G\sigma\ \ (watts/m^2) \tag{2.20}$$

Now to analyze the reception part, all the antennas have a parameter called effective area, which is the ratio of the power available in the antenna receiving signal to the power density of the wave:

$$A_e = \frac{P_a}{< |\vec{S}| >} \tag{2.21}$$

To determine the relationship between the effective area and gain, it is going to be considered the system showed in the following figure.

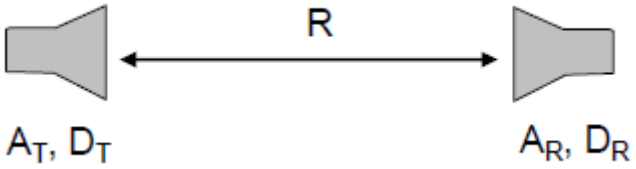


Figure 2.1. Link of two antennas.

Suppose two antennas with directivity D_T, D_R and effective area A_T, A_R respectively separated by distance R.
As it was shown before, if the transmitting antenna has a directivity D_T, the power density at a distance R is as the equation (2.15) :

$$< |\vec{S}_T| >= \frac{P_T}{4\pi R^2} D_T \tag{2.22}$$

Then the received power by the receiving antenna is:

$$P_R =< |\vec{S}_T| >\cdot A_R = \frac{P_T}{4\pi R^2} D_T A_R \tag{2.23}$$

The same expression can be shown as:

$$D_T A_R = \frac{P_R}{P_T} 4\pi R^2 \tag{2.24}$$

By the reciprocity theorem in electromagnetics it is known that the antennas have the same behavior in both transmitting and receiving. For that reason the transmitting and receiving antenna can be exchanged obtaining the following expression:

$$D_R A_T = \frac{P_R}{P_T} 4\pi R^2 \tag{2.25}$$

Relating the two last equations (2.24) and (2.25), yields:

$$D_T A_R = D_R A_T \;\Rightarrow\; \frac{D_T}{A_T} = \frac{D_R}{A_R} \tag{2.26}$$

Once it is known the relationship between directivity and effective area in a link of two antennas we are going to obtain the effective area of a general type of antenna considering a simple case. Suppose that the transmitting antenna is isotropic and the receiving antenna is an ideal dipole:

$$D_T = 1 \;\Rightarrow\; A_T = \frac{A_R}{D_R} \tag{2.27}$$

The far-fields radiated by an ideal dipole are obtained by calculating the magnetic vector potential and deriving them by Maxwell's equations. The derivation can be seen in the appendix A (A.3).

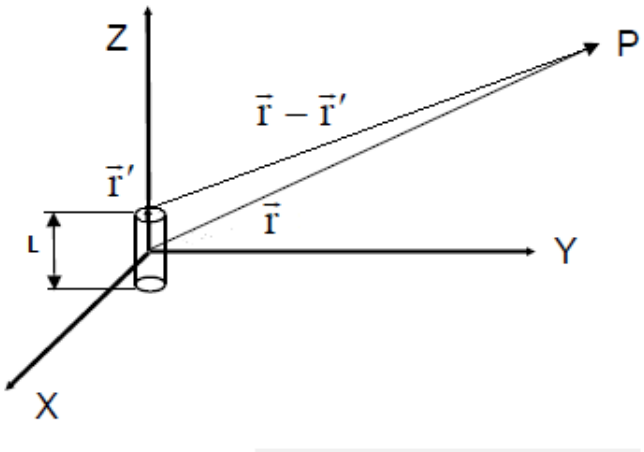


Figure 2.2. Ideal dipole placed at the origin.

$$\vec{A}(\bar{r}) = \frac{\mu}{4\pi} \iiint \frac{\vec{J}(\bar{r}')}{|\bar{r} - \bar{r}'|} e^{ik|\bar{r}-\bar{r}'|}\, dV' \tag{2.28}$$

As the dipole is ideal, the current density is considered constant along its length:

$$\vec{J}(\bar{r}') = I\delta(x')\delta(y')\hat{z}, \qquad -\frac{L}{2} < z' < \frac{L}{2} \tag{2.29}$$

The condition of short (ideal) dipole is given by the following relation:

$$L \ll \lambda \tag{2.30}$$

Approximations for far-field:

- Amplitude:

$$|\bar{r} - \bar{r}'| \simeq |\bar{r}| = r \tag{2.31}$$

- Phase:

$$k|\bar{r} - \bar{r}'| \tag{2.32}$$

Using trigonometric relations from the triangle formed by the origin, the observation point and the extreme of the ideal dipole, a better approximation can be obtained.

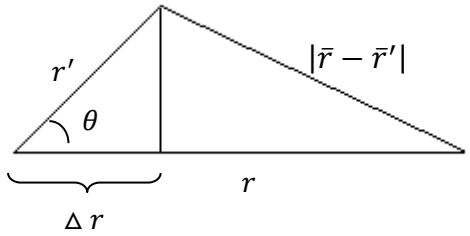


Figure 2.3. Trigonometric scheme of the ideal dipole radiation.

$$cos\theta = \frac{\Delta r}{r'} \;\Rightarrow\; \Delta r = r' cos\theta \tag{2.33}$$

$$cos\theta = \hat{r} \cdot \hat{r}' = \hat{r}' \cdot \hat{r} \tag{2.34}$$

$$\Delta r = r'\hat{r}' \cdot \hat{r} = \bar{r}' \cdot \hat{r} \tag{2.35}$$

Therefore the distance $|\bar{r} - \bar{r}'|$ can be approximated to the following equation:

$$|\bar{r} - \bar{r}'| \simeq r - \Delta r = r - \bar{r}' \cdot \hat{r} \tag{2.36}$$

Using this last approximation the following expression for the phase is obtained:

$$k|\bar{r} - \bar{r}'| \simeq k(r - \Delta r) = \frac{2\pi}{\lambda}(r - \Delta r) \tag{2.37}$$

As $\Delta r \propto |\bar{r}'|$ and $L << \lambda$, then $\Delta r \ll \lambda$. The final approximated expression of the phase can be written as:

$$k|\bar{r} - \bar{r}'| \simeq \frac{2\pi}{\lambda}(r - \Delta r) \simeq \frac{2\pi}{\lambda} r = kr \tag{2.38}$$

The expression of the magnetic vector potential becomes:

$$\vec{A}(\bar{r}) = \frac{\mu}{4\pi r} e^{ikr} \int_{-\frac{L}{2}}^{\frac{L}{2}} I\, dz' \hat{z} = \frac{\mu IL}{4\pi r} e^{ikr} \hat{z} \tag{2.39}$$

Then the expressions can be easily derived by the equations described in A.3 (appendix A).

$$\vec{A} = A_z \hat{z} = A_z cos\theta \hat{r} - A_z sin\theta \hat{\theta} = A_r \hat{r} + A_\theta \hat{\theta} \tag{2.40}$$

Where:

$$A_r = A_r(r, \theta) = \frac{\mu IL}{4\pi r} e^{ikr} cos\theta \tag{2.41}$$

$$A_\theta = A_\theta(r, \theta) = -\frac{\mu IL}{4\pi r} e^{ikr} sin\theta \tag{2.42}$$

We are interested in the electric and magnetic fields, so that to obtain the fields the equations in A.3 (appendix A) have to be used:

$$\vec{H} = \frac{1}{\mu} \nabla \times \vec{A} = \frac{1}{\mu} \frac{1}{r} \left[\frac{\partial}{\partial r} (r A_\theta) - \frac{\partial A_r}{\partial \theta} \right] \hat{\phi} = \frac{IL}{4\pi} k^2 sin\theta \left[\frac{1}{ikr} - \frac{1}{(ikr)^2} \right] e^{ikr} \hat{\phi} \tag{2.43}$$

$$\vec{E} = -\frac{\nabla \times \vec{H}}{i\omega\epsilon} = E_r \hat{r} + E_\theta \hat{\theta} \tag{2.44}$$

Where:

$$E_r = \frac{IL}{4\pi} \eta k^2 2cos\theta \left[-\frac{1}{(ikr)^2} + \frac{1}{(ikr)^3} \right] e^{ikr} \tag{2.45}$$

$$E_\theta = \frac{IL}{4\pi} \eta k^2 sin\theta \left[\frac{1}{ikr} - \frac{1}{(ikr)^2} + \frac{1}{(ikr)^3} \right] e^{ikr} \tag{2.46}$$

$$\eta = \sqrt{\frac{\mu}{\epsilon}} \tag{2.47}$$

$\eta \equiv$Intrinsic impedance of the medium. For free space:

$$\eta_0 = \sqrt{\frac{\mu_0}{\epsilon_0}} = 120\pi\ \Omega \tag{2.48}$$

The expressions of fields contain both the induced and the far-field. Out of all the terms of the equations (2.43) and (2.44), only the far field is considered important. Hence, the near field is neglected. The conditions for far field are the following:

$$r \gg \lambda \tag{2.49}$$

$$kr = \frac{2\pi}{\lambda} r \gg 1 \tag{2.50}$$

Therefore it can be noted that:

$$\frac{1}{ikr} \gg \frac{1}{(ikr)^2} \gg \frac{1}{(ikr)^3} \tag{2.51}$$

Taking into account the equation (2.51), the expressions for far field radiated by an ideal dipole can be approximated to:

$$\vec{H} = \frac{IL}{4\pi} \frac{k^2}{ik} \frac{e^{ikr}}{r} sin\theta \hat{\phi} \tag{2.52}$$

$$E_r \simeq 0 \;\; \Rightarrow \;\; \vec{E} = \frac{IL}{4\pi} \eta \frac{k^2}{ik} \frac{e^{ikr}}{r} sin\theta \hat{\theta} = \eta H_\phi \hat{\theta} \tag{2.53}$$

Hence, the far field wave behaves as a plain wave, although it is a spherical wave.

It is useful to define the radiation pattern as the ratio of the magnitude of the electric field to the maximum magnitude value of the electric field:

$$F(\theta, \phi) = \frac{E(\theta, \phi)}{E_{max}} \tag{2.54}$$

For the case of the ideal dipole the radiation pattern is:

$$F(\theta, \phi) = sin\theta \tag{2.55}$$

Once the expression of far fields radiated of an ideal dipole, then the next step is to obtain the directivity. As mentioned before in the equation (5.10), directivity is the ratio of the radiation intensity in a certain direction to the radiation intensity of an isotropic antenna.

$$D = D_{max}(\theta, \phi) = \frac{U_m}{U_{iso}} = \frac{U_m}{P_r} 4\pi \tag{2.56}$$

Where U_m is the maximum value of the radiation intensity and it is independent on θ, ϕ.

$$U(\theta,\phi) = \frac{|E(\theta,\phi)|^2}{2\eta} = U_m|F(\theta,\phi)|^2 \tag{2.57}$$

The radiated power can be expressed as follows:

$$P_r = \iint U(\theta,\phi)d\Omega = U_m \iint |F(\theta,\phi)|^2 d\Omega = U_m\Omega_A \tag{2.58}$$

The directivity expression is simplified to:

$$D = \frac{U_m}{P_r}4\pi = \frac{U_m}{U_m \iint F(\theta,\phi)d\Omega}4\pi = \frac{4\pi}{\iint |F(\theta,\phi)|^2 d\Omega} = \frac{4\pi}{\Omega_A} \tag{2.59}$$

Where Ω_A is called the beam solid angle. For an ideal dipole the beam angle is:

$$\Omega_A = \iint |F(\theta,\phi)|^2 d\Omega = \int_{\phi=0}^{2\pi}\int_{\theta=0}^{\pi} sin^3\theta d\theta d\phi = 2\pi \int_{\theta=0}^{\pi} sin^3\theta d\theta = 2\pi\frac{4}{3} \tag{2.60}$$

Therefore the directivity of the ideal dipole is:

$$D = \frac{4\pi}{2\pi\frac{4}{3}} = \frac{3}{2} \tag{2.61}$$

We recall that all these previous analysis of the ideal dipole has been made to get the effective area of an isotropic antenna. In order to obtain this relation it has only got the effective area of the ideal dipole. This is not intuitive and some more concepts have to be mentioned.

A receiving antenna can be seen as a voltage source transmitting power to a load using a transmission line as a waveguide.

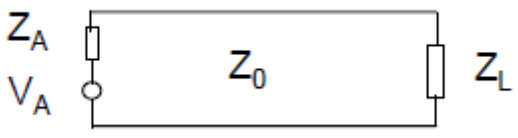


Figure 2.4. Equivalent circuit of an ideal dipole receiving signal.

Where:

Z_A ≡Antenna impedance. $Z_A = R_A + jX_A$
R_A ≡Antenna resistance. $R_A = R_r + R_{ohm}$
R_r ≡Radiation resistance
R_{ohm} ≡Ohmic losses resistance
X_A ≡Antenna reactance
Z_L ≡Load impedance. $Z_L = R_L + jX_L$

The current of the equivalent circuit can be expressed as:

$$I = \frac{V_A}{Z_T} = \frac{V_A}{R_A + R_{ohm} + R_L + j(X_A + X_L)} \tag{2.62}$$

Taking the magnitude of the current:

$$|I| = \frac{|V_A|}{[(R_A + R_{ohm} + R_L)^2 + (X_A + X_L)^2]^{1/2}} \tag{2.63}$$

The receiving power by the load is:

$$P_L = \frac{1}{2}|I|^2 R_L = \frac{|V_A|^2}{2} \frac{R_L}{(R_A + R_{ohm} + R_L)^2 + (X_A + X_L)^2} \tag{2.64}$$

The maximum power transfer theorem establishes that the power transmitted to the load is maximum if:

$$Z_L = {}^*Z_A \tag{2.65}$$

That establishes:

$$R_L = R_A = R_r + R_{ohm} \tag{2.66}$$

$$X_L = -X_A \tag{2.67}$$

Considering the case of maximum power transfer, the power received by the load becomes:

$$P_L = \frac{|V_A|^2}{8} \frac{1}{R_r + R_{ohm}} \tag{2.65}$$

In addition if the antenna does not have losses:

$$R_{ohm} = 0 \;\Rightarrow\; P_L = \frac{|V_A|^2}{8}\frac{1}{R_r} \tag{2.66}$$

In general the antenna is matched with the transmission line guiding the received wave to the load. That means the antenna impedance is equal to the characteristic impedance of the transmission line.

$$Z_A = Z_0 \tag{2.67}$$

The maximum power that can be delivered to the load corresponds with the available power at the receiving antenna:

$$P_a = \frac{1}{8}\frac{|V_A|^2}{R_A} = \frac{1}{8}\frac{|V_A|^2}{Z_0} \tag{2.68}$$

Therefore considering the power reflected at the load due to mismatch, the relationship between the power available and the power delivered to the load is:

$$P_L = P_a(1 - |\rho_L|^2) \tag{2.69}$$

Where ρ_L is the reflection coefficient at the load:

$$\rho_L = \frac{Z_L - Z_0}{Z_L + Z_0} \tag{2.70}$$

Matching the load with the transmission line:

$$Z_L = Z_0 \;\Rightarrow\; \rho_L = 0 \;\Rightarrow\; P_L = P_a \tag{2.71}$$

After defining the parameter of a receiving antenna, we are going to obtain the effective area of the ideal dipole. First of all we consider that the ideal dipole does not have any losses ($R_{ohm} = 0$). As it was mentioned before the effective area is:

$$A_e = \frac{P_a}{< |\vec{S}| >} = \frac{|V_A|^2}{8}\frac{1}{R_r}\frac{1}{< |\vec{S}| >} \tag{2.72}$$

Where:

$$|V_A| = |E|L \tag{2.73}$$

$$< |\vec{S}| >= \frac{|E|^2}{2\eta} \tag{2.74}$$

$$P_r = U_m \iint |F(\theta,\phi)|^2 d\Omega = \iint \frac{1}{2}\mathbb{R}e(E \times H^*)\cdot \overrightarrow{dS} = \frac{1}{2}\eta\left(\frac{kIL}{4\pi}\right)^2 \iint sin^3\theta d\theta d\phi \tag{2.75}$$

$$U_m = \frac{1}{2}\eta\left(\frac{kIL}{4\pi}\right)^2 \tag{2.76}$$

$$R_r = \frac{2P_r}{|I|^2} = \frac{2U_m\Omega_A}{|I|^2} = 2\frac{\frac{1}{2}\eta\left(\frac{kIL}{4\pi}\right)^2 2\pi\frac{4}{3}}{|I|^2} = \eta\left(\frac{kL}{4\pi}\right)^2 2\pi\frac{4}{3} = \eta\left(\frac{L}{\lambda}\right)^2\frac{2\pi}{3} \tag{2.77}$$

Then substituting all these terms in the equation of the effective area we obtain:

$$A_e = \frac{|V_A|^2}{8}\frac{1}{R_r < |\vec{S}| >} = \frac{|E|^2L^2}{8}\frac{1}{\eta\left(\frac{L}{\lambda}\right)^2\frac{2\pi}{3}\frac{|E|^2}{2\eta}} = \frac{3\lambda^2}{8\pi}$$

Once the ideal dipole has been analyzed, the expression of the effective area of an isotropic antenna can be obtained. It is important to remember the expression of the effective area of an arbitrary antenna, obtained considering a link between an isotropic antenna as transmitting antenna and an ideal dipole as receiving antenna.

$$A_T = \frac{A_R}{D_R} = \frac{\frac{3\lambda^2}{8\pi}}{3/2} = \frac{\lambda^2}{4\pi} \tag{2.78}$$

If it is considered again a two link antenna with an isotropic antenna as the transmitting antenna ($D_T = 1$) and a general type of antenna as the receiving antenna:

$$\frac{D_T}{A_T} = \frac{D_R}{A_R} \;\Rightarrow\; A_R = A_T D_R = \frac{\lambda^2}{4\pi}D_R \tag{2.79}$$

If we consider the efficiency of the antenna then the effective area of a general antenna can be written as follows:

$$A_e = \frac{\lambda^2}{4\pi}G \tag{2.80}$$

Finally to obtain the received power (P_r) by the radar, we have to multiply the effective area of the antenna by the power density incoming given by the equation (2.20):

$$P_r = A_e < |\vec{S}_r| >= \frac{P_t}{(4\pi R^2)^2}G\frac{\lambda^2}{4\pi}G = P_t\frac{\lambda^2G^2\sigma}{(4\pi)^3R^4} \tag{2.81}$$

3. PHASED ARRAY RADAR

Once the most basic concepts of a radar have been defined, this project is focused on a particular type of radar called phased array radar with eight microstrip dipoles working at a frequency of 2.4 GHz (S band) with a spacing element of $\lambda/2$. Phased array radar is characterized by its antenna. The antenna is a phased linear array of antennas, a set of multiple radiation elements arranged as a straight line, in which the radiation pattern can be reinforced in a particular direction and suppressed in undesired directions. A phased array can provide narrow directive beams that may be steered electronically as shown in the figure 3.1. This is the main advantage of the phase array radar because simply using different phases to the current supplying each array element, the directivity varies. The antenna radar can radiate the electromagnetic power, used to detect targets, in different directions obviating the needed of any mechanical rotation. In addition a phased array can also scan a beam at high electronic speeds and can even have multiple simultaneous main beams.

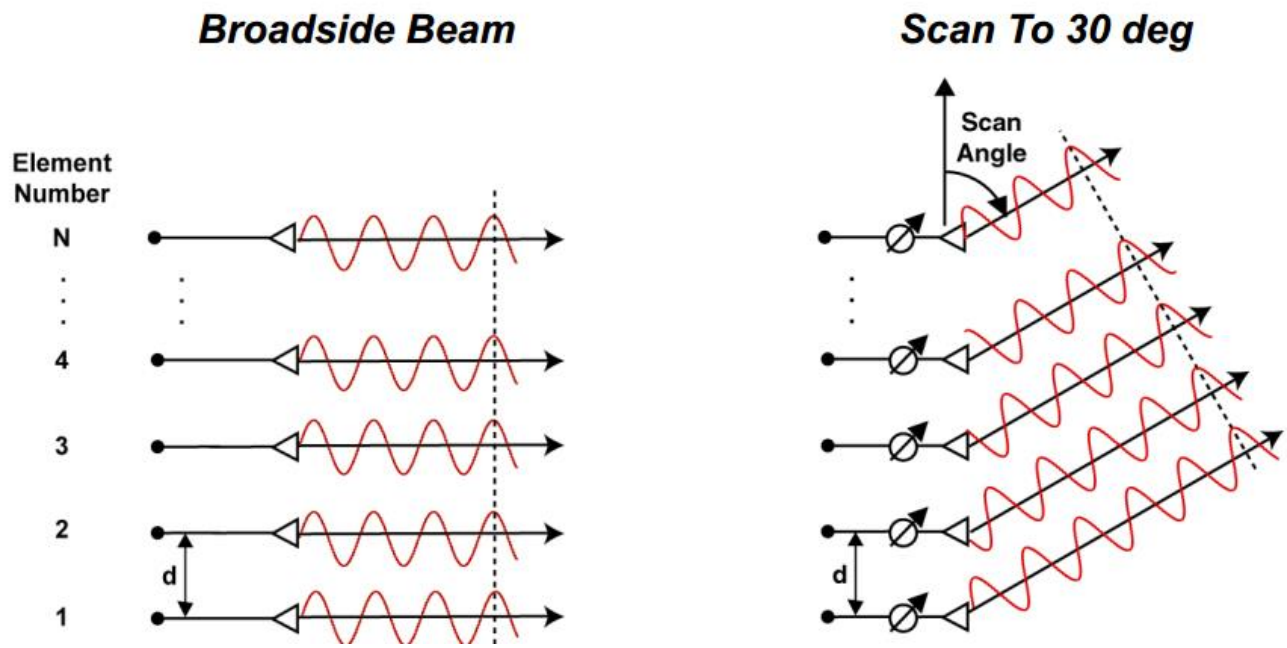


Figure 3.1. Different scan angles of a phased array antenna.

The basic scheme of the phased array antenna will be used to transmit the RF signal is showed in the following figure.

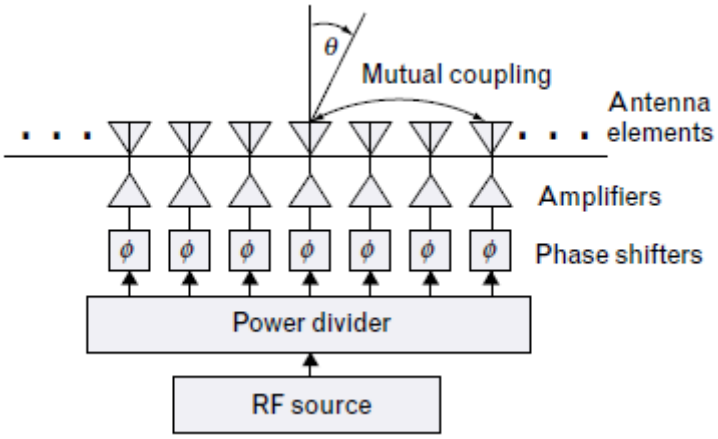


Figure 3.2. Scheme of a general phased array antenna with infinite elements.

The transmitter is compound of a RF source, which generates the radio-frequency signal. After that the power is divided equally to supply each array element with the same amount of power. Phase shifters are responsible for establishing a phase shift in the current supplied to each array element, in order to change the directivity of the antenna. Finally the signal is amplified to be sent and reach targets at long ranges.

An N-element phased array is formed by N identical radiating antennas separated by a distance d along an axis. In the figure above it can be noted that depending on the phase shifts of each element array the interferences of the signal will be constructive in a certain scan angle and destructive in others.

The principle of this type of radar is developed considering the receiver case for simplicity, while similar concepts are held for transmitter case as well due to the reciprocity theorem applied in antennas. Variable time delays are incorporated at each signal path to control the phases of the signals before combining all the signals together at the output. A plain wave-beam is assumed to be incident upon the antenna array at an angle of θ to the normal direction. Due to the spacing between array elements the beam will experience a time delay in reaching successive antennas, corresponding to the following equation.

$$\Delta t = \frac{2\pi d sin\theta}{\lambda} \tag{3.1}$$

Hence if the incident beam is a time-harmonic signal at a frequency f with amplitude of A, the signals received by each of the array elements can be written as:

$$S_i = Ae^{-jn\Delta t} \tag{3.2}$$

The plane wave incident at an angle upon the phased array experiences a linear delay progression at the successive antenna elements. Therefore the variable delay circuits must be set to a similar but with reverse delay progression to compensate for the delay of the signal arrived at the antenna elements. In linear arrays, variable time delays are designed to provide uniform phase progression across the array. Therefore the signal in each channel at the output of the variable delay block can be written as:

$$S_i' = e^{-jn\Delta t} e^{jn\Delta\alpha} \tag{3.3}$$

In this equation, α denotes the phase difference provided by two successive variable time delay blocks. Therefore the array factor which is equal to the sum of all the signals normalized to the signal at one path can be represented as follows:

$$AF = \sum_{i=1}^{N} e^{-jn(\Delta t - \Delta\alpha)} \tag{3.4}$$

According to equation 3.4, the peak of the array factor occurs at an incident angle which can be determined by:

$$\Delta\alpha = \frac{2\pi d sin\theta}{\lambda} \tag{3.5}$$

$$\theta = arcsin\left(\frac{\Delta\alpha}{2\pi d}\right) \tag{3.6}$$

This angle is called the scan angle. At the scan angle the linear delay progression experienced by the wave arriving at the successive antennas is compensated with the time delays incorporated at each path resulting in constructive interference at the output of the receive array. Therefore the array factor also can be written as the following equation:

$$AF = \frac{sin^2\left[\frac{N}{2}\left(\frac{2\pi d}{\lambda} sin\theta_{in} - \Delta\alpha\right)\right]}{sin^2\left[\left(\frac{2\pi d}{\lambda} sin\theta_{in} - \Delta\alpha\right)\right]} \tag{3.7}$$

At the scan angle "θ", the array factor has a maximum value of N^2. For incident angles different from the scan angle, the array factor will be lower than the value at the equation (3.7), indicating spatial selectivity of phased array. In addition to the spatial filtering, one of the main capabilities

of phased array is that the peak gain of the array, according to equation (3.6), can be controlled by electronically tuning the variable time delays eliminating the need for any mechanical system which rotates the antenna array.

It should be noted that the advantages of using a phased array is increased as the number of elements in the phased array is increased. As mentioned before the maximum value of the array factor N^2 is directly proportional to the number of array elements. Furthermore, the beam width of the array can be decreased by increasing the number of array elements in order to enhance the spatial selectivity of phased array. The beam width can be also decreased by increasing the element spacing, but element spacing greater than half-wavelength results in undesired grating lobes.

4. FEED AND BIAS NETWORK OF A PHASED ARRAY RADAR

Phased array antennas are usually composed of a feed network and a bias network which allows the operation of the antenna working together. Feed networks are used to distribute the radio-frequency output signal of the transmitter to the radiating elements. There are many ways to feed the arrays. In general array feed networks can be classified into three basic categories: constrained feed, space feed and semi-constrained feed which is a hybrid of the constrained and the space feeds. In a space feed network, the array is usually illuminated by a separate feed horn located at an appropriate distance from the array. Due to the space of the free space between the feed and the radiating elements, this type of feed networks is not a good candidate for planar arrays. This restricts the applications for this type of feed network. The constrained feed, which is usually the simplest method of feeding an array, generally consist of a network which receives the power from an RF source and distributes it to the antenna elements with a feed line and passive elements such as power dividers also known as couplers. The constrain feed itself can be categorized into two basic types: parallel feeds and series feeds. The architectures based on these two feed networks are the most common approach to design phased arrays. The next stage of the phased array is the phase shifters network. Phase shifters can be placed at any stage of phased array. The most common architecture places the phase shifters connecting feed network outputs with each radiating element. Depending on a certain DC control voltage applied to each phase shifter, it produces a certain phase shift at its feed RF signal. Bias network is responsible for controlling automatically these control voltages to form a beam in the desired direction.

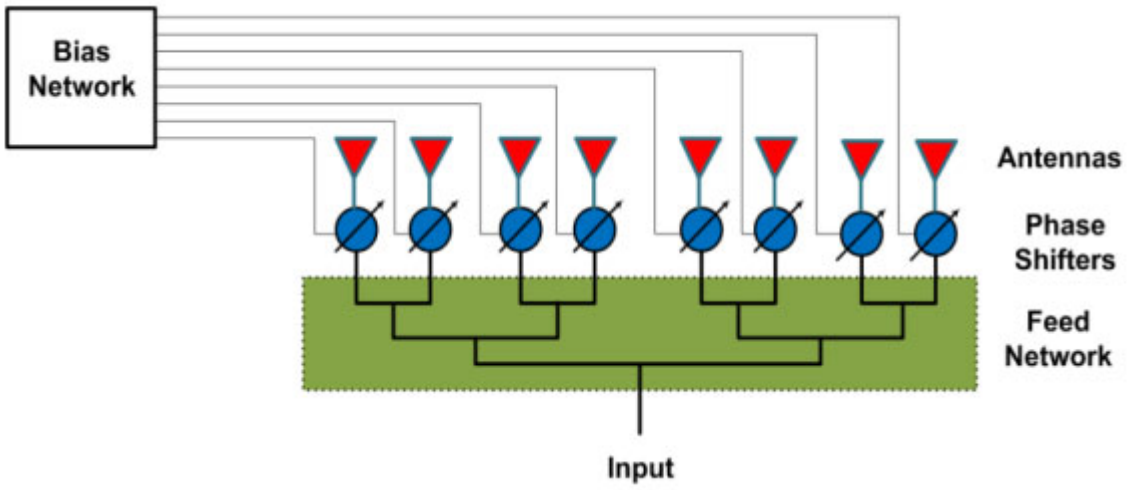


Figure 4.1. Scheme of a phased array using a parallel feed network.

In the figure above a scheme of the feed and bias network is shown.

4.1. Feed Network

As mentioned before space feed networks are not suitable for planar arrays, this limits some application for this type of phased array. For the purpose of this project it is going to be considered only constrain feed network, which in turn, is divided into two types: parallel and series feeds.

- Series-fed array: In a series-fed array the input signal, fed from one end of the feed network, is coupled serially to the radiating elements of the array as shown in the following figure.

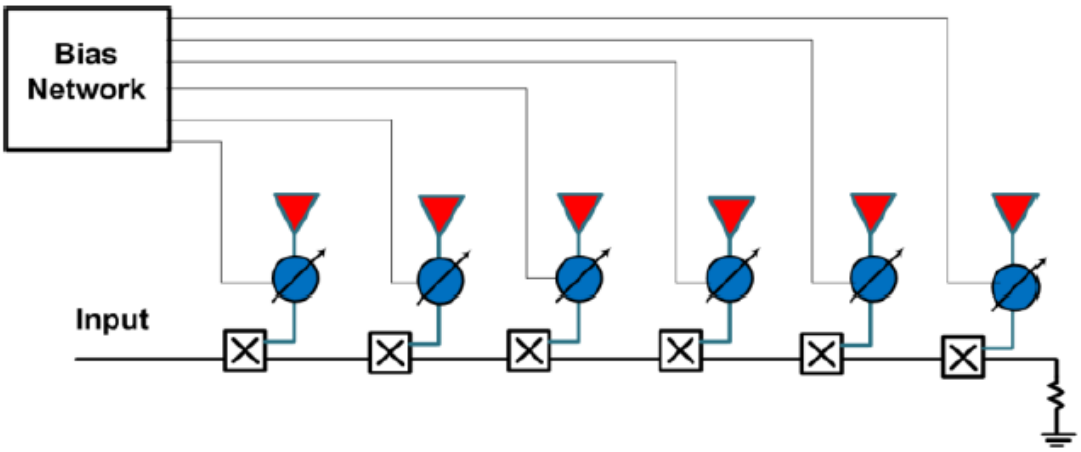


Figure 4.2. Scheme of a series-fed array.

The main advantage of the series-fed array is the compactness. Beside compactness, the small size of the series-fed arrays results in less insertion losses by the feed network. In addition a series-fed array with N elements requires less phase shifters than the parallel-fed arrays. However the cumulative nature of phase shift through the feed network results in an increased beam squint versus frequency, which is one of the main limitations in series-fed designs. The loss through the phase shifters is also cumulative in series-fed arrays which can be an issue in the design of arrays with a large number of array elements.

- Parallel-fed array: Also called corporate feeds. In a parallel-fed array the input signal is divided in a corporate tree network to all the radiating elements of the array antenna, as shown in the figure 4.1. These types of networks typically employ power dividers. Their performance depends on the architecture of the power dividers used.

Considering the inconveniences given by the series-fed array, a parallel-fed array is chosen for the feed network design developed in the next chapter.

4.2. Bias Network

Phase array radars make continuously a scan angle sweep to detect targets at different positions. As mentioned the directivity of the antenna depends on the phase of the signal which supplies each of the radiating element. In order to change continuously the directivity, phase shifters are responsible for providing different phase shifts at the incoming signal. Phase shifter is a microwave device which provides different phase shifts which are controlled by a control DC voltage. Therefore an automatic system which provides different continuous voltages to the phase shifters is needed. This automatic system can be implemented using a micro-controller connected to a computer establishing a bidirectional communication between them. A software running in the computer could generate different scan angles. Based on a measurement pattern and given the scan angle, control voltages of each phase shift can be obtained to provide the desired scan angle or directivity.

5. PARALLEL-FED NETWORK DESIGN

In this chapter the feed network of the array antenna is going to be designed. As mentioned, a parallel-fed array is chosen for this purpose. The array antenna of the phased array radar has eight radiating elements. Therefore using several power dividers the network should provide eight outputs with the same power level, one for each radiating element, from one unique input. The parallel-fed array can be implemented using a tree architecture of power dividers made up of one input port and two output ports as shown in the following figure.

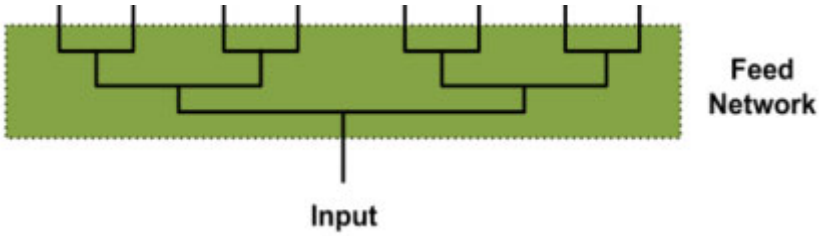


Figure 5.1. Scheme of the parallel-fed array of eight radiating elements.

There are several types of power dividers, but for the specified feed network the 3 dB/0º Wilkinson is the most suitable. This device is a three-port network composed of one input port and two output ports. The power supplied at the input is equally split. The outputs provide half of the input power supplied. The outputs are isolated, and there is no phase shift between them. This type of power divider is often implemented using microstrip lines as depicted in the following figure.

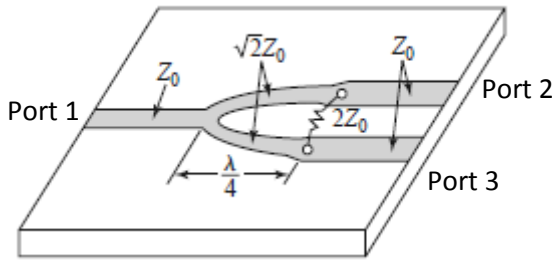


Figure 5.2. Microstrip 3 dB/0º Wilkinson coupler.

For the parallel-fed network chosen at least seven power dividers are needed. The architecture of the feed network can be divided into three stages. Each stage uses different designs of Wilkinson power divider, as can be seen in the following figure.

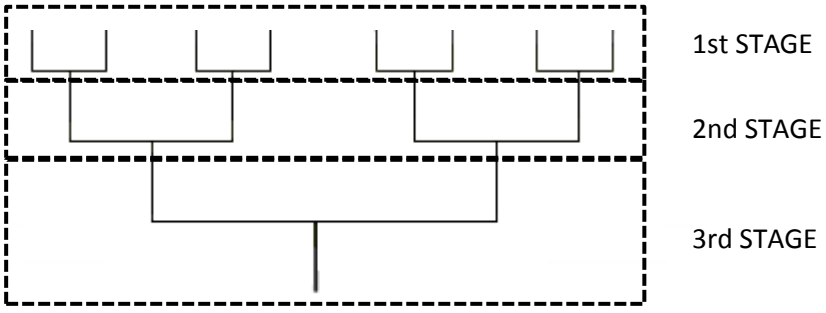


Figure 5.3. Archiitecture of the parallel-fed network.

The first stage is composed by four power dividers. This results in eight output ports where the dipoles are connected. As it was defined in the chapter 3, the separation between array elements is $\lambda/2$. The outputs of the feed network should be separated by a distance of $\lambda/2$. As the design frequency is 2.4 GHz, the distance between radiating elements is 6.25 cm. Therefore the power dividers at the second and third stage depend on the physical design of the first stage. For this reason, first of all the power dividers of the first stage will be designed. After that, taking into account the element spacing between radiating elements in the array antenna, the power dividers of the second and third stage will be designed. The idea is designing and simulating the power dividers of each stage individually and then simulate the complete feed network. Before designing the power dividers required for the parallel-fed array is important to recall the concept of the scattering parameters of a network. Any radio-frequency device is characterized by its S parameters (Scattering).

5.1. S parameters of the 3 dB/0º Wilkinson

S parameters of a radio-frequency device are defined in function of the incident and reflected power at its ports. The equations for the scattering parameters are the following:

$$b_1 = a_1 s_{11} + a_2 s_{12} + a_3 s_{13} \tag{5.1.1}$$

$$b_2 = a_1 s_{21} + a_2 s_{22} + a_3 s_{23} \tag{5.1.2}$$

$$b_3 = a_1 s_{31} + a_2 s_{32} + a_3 s_{33} \tag{5.1.3}$$

They can also be expressed using matrix form:

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{5.1.4}$$

Where:

$$a_i = \frac{V_i^+}{\sqrt{Z_0}} \tag{5.1.5}$$

$$b_i = \frac{V_i^-}{\sqrt{Z_0}} \tag{5.1.6}$$

a_i ≡Normalized incident power at the port i of the device.
b_i ≡Normalized reflected power at the port i of the device.
V_i^+ ≡Incident voltage wave at the port i.
V_i^- ≡Reflected voltage wave at the port i.

The physical meaning of the scattering parameters is the following:

- Reflection coefficient at the port k with the other ports matched:

$$s_{kk} = \left.\frac{b_k}{a_k}\right|_{The\ other\ ports\ different\ from\ k\ are\ matched} \tag{5.1.7}$$

- Gain seen at the port k if the port j is stimulated by a generator, with the other ports different from j matched:

$$s_{kj} = \left.\frac{b_k}{a_j}\right|_{The\ other\ ports\ different\ from\ j\ are\ matched} \tag{5.1.8}$$

Once the physical meaning of the S parameters has been defined, the next equation shows the S parameters of a 3 dB/0º Wilkinson power divider of the figure 5.2.

$$S = \frac{-j}{\sqrt{2}}\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{5.1.9}$$

The behavior of the 3 dB/0º Wilkinson can be deduced by its scattering parameter matrix:

- The reflection coefficient at a given port is zero, when the other ports are matched:

$$s_{11} = s_{22} = s_{33} = 0 \tag{5.1.10}$$

- The ports 2 and 3 also known as the output ports are isolated:

$$s_{23} = s_{32} = 0 \tag{5.1.11}$$

- The power at the output ports of the device is one-half of the input power. The output signal of each port are in phase:

$$s_{21} = s_{31} = s_{12} = s_{13} = \frac{-j}{\sqrt{2}} \tag{5.1.12}$$

The factor j reflects the phase difference between the input and output signal, due to the line of length $\lambda/4$ which results in an electrical length of 90 degrees.
If the incident power and reflected power at a general port i are defined as follows:

$$P_{INi} = \frac{1}{2}|a_i|^2 \tag{5.1.13}$$

$$P_{REFi} = \frac{1}{2}|b_i|^2 \tag{5.1.14}$$

Therefore the scattering parameter s_{21} can be obtained from the equation (5.1.2) as:

$$s_{21} = \frac{b_2}{a_1}\Big|_{a_2=0, a_3=0} \tag{5.1.15}$$

Taking the magnitude the following relationship can be obtained:

$$|s_{21}|^2 = \frac{1}{2} = \frac{|b_2|^2}{|a_1|^2} = \frac{P_{REF2}}{P_{IN1}} \tag{5.1.16}$$

The ratio of the reflected power at the port two to the incident power at the port one is one half. The result is obtained between the third port and the first one as establish equation (5.1.12).

5.2. Design and simulation of a 3 dB/0º Wilkinson using AWR

In this section it is going to be designed the above described power divider using the radio frequency software AWRDE 10. Before designing the power divider it is important to define some design characteristics such as the operation frequency and the substrate used:

- Design frequency: $f = 2.4\ GHz$ (S band).
- Substrate characteristics:
  - Relative permittivity: $\epsilon_r = 4.5$

- Dielectric thickness: $H = 1.5\ mm$
- Conductor thickness: $t = 35\ \mu m$
- Loss tangent: $tan\delta = 0.01$
- Electrical resistance: $RHO = 0.77$
  This parameter is the electrical resistance of the conductor normalized with respect to the electrical resistance of the gold. The electrical resistance at 20-25 degrees Celsius is:

$$Rho_{Au} = 2.22 \cdot 10^{-8}\ \Omega m \tag{5.2.1}$$

$$Rho_{Cu} = 1.71 \cdot 10^{-8}\ \Omega m \tag{5.2.2}$$

Therefore the value of RHO is obtained from the following equation:

$$RHO = \frac{Rho_{Cu}}{Rho_{Au}} = 0.77027 \tag{5.2.3}$$

Once the basic design characteristics are defined, the design can be started. First of all the ideal case is simulated. It consists of using ideal transmission lines, loss-less, to implement the tracks and ports of the power divider. This gives a first approach of the final design solution. After that in order to obtain a more accurate design, real transmission lines can be used in the design. These real transmission lines are lossy lines whose characteristic parameters are defined by its substrate. Therefore, in order to finish the analysis, the physical schematic is obtained and the circuit is implemented. Finally some measures are taken to ensure the well operation of the power divider.

5.2.1. Ideal 3 dB/0º Wilkinson

As mentioned before, the ideal simulation consists on a first approach solution design using loss-less transmission lines. It only produces a phase difference. The schematic of the power divider is shown in the figure 5.4.

The characteristic impedance of the ports and line transmissions used is 50 Ω. The values of the characteristic impedance of the tracks and resistance match with the values from the figure 5.2.
The response of the power divider is viewed using a frequency sweep from 2 to 2.8 GHz. The scattering parameters are shown in the figure 5.5.

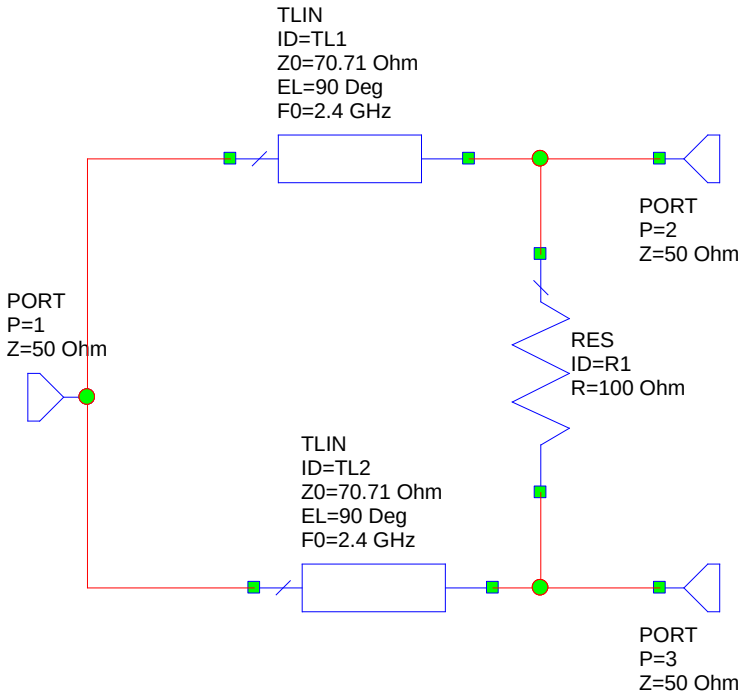


Figure 5.4. Schematic of an ideal 3 dB/0º Wilkinson.

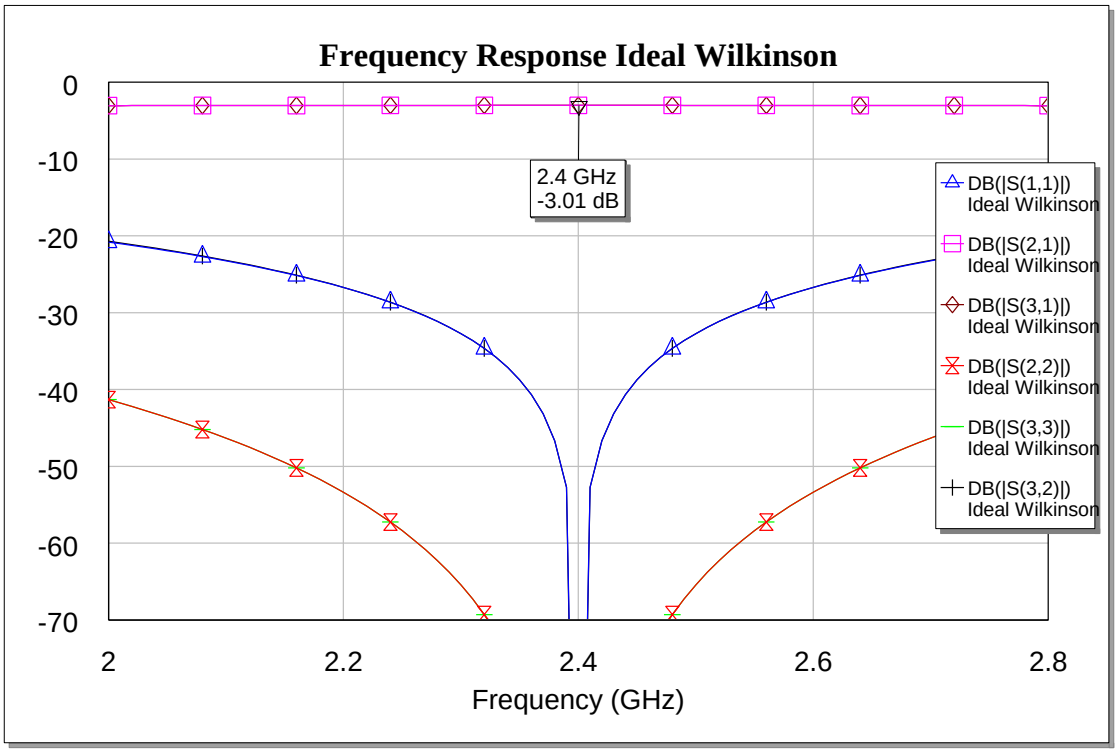


Figure 5.5. Frequency response of the ideal 3 dB/0º Wilkinson.

The figure above shows the different values of the scattering parameters in dB at different frequencies. As it can be noted at the operation frequency:

$$S_{11}(dB) = S_{22}(dB) = -\infty = 10log|s_{11}|^2 = 10log|s_{22}|^2 \quad (5.2.1.1)$$

Therefore:

$$s_{11} = s_{22} = 0 \quad (5.2.1.2)$$

Also the relation between the input and output ports can be seen in the following scattering parameters in dB:

$$S_{21}(dB) = -3.01 = 10log|s_{21}|^2 \quad (5.2.1.3)$$

$$S_{31}(dB) = -3.01 = 10log|s_{31}|^2 \quad (5.2.1.4)$$

Therefore:

$$|s_{21}|^2 = |s_{31}|^2 = 10^{-0.301} \simeq \frac{1}{2} \quad (5.2.1.5)$$

This means that the power at the output ports is one-half of the power at the input port as it is desired. The isolation between the output ports can be seen analyzing the parameter s_{32} or s_{23}, which are close to minus infinite.

5.2.2. Simulation of a real 3 dB/0º Wilkinson

Once a first approach has been obtained, the next step is to simulate the 3 dB/0º Wilkinson using microstrip lines. In order to obtain accurate results the substrate used has to be completely characterized. Using AWRDE 10 software, the substrate can be characterized using the element in the following figure:

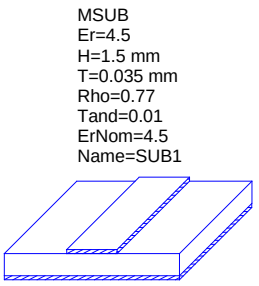


Figure 5.6. Substrate defining the parameters of the microstrip line.

After defining the substrate it is used to implement the power divider, the lines have to be defined. Microstrip lines have two design parameters: physical length and width. These two parameters are set by the tool TXLine that calculates the physical length and the width of each line using several input parameters such as frequency, type of conductor, impedance, electrical length, dielectric constant and so on. These values match whit those of the figure 5.2. The conductor of the substrate used is copper.

As there are lines with different width, there will be some parasitic effects. These effects should be taken into account using the element MTEE shown in the following figure.

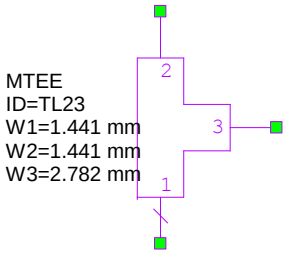


Figure 5.7. Microstrip tee junction.

Another effect to take into account is the parasitic inductance present at the resistance even using SMD components. This effect it can be included in the simulation using the element in the following figure.

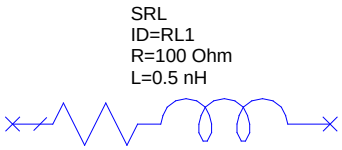


Figure 5.8. Resistance with parasitic inductance.

The schematic of the real 3 dB/0º Wilkinson is shown in the figure 5.9. This schematic it is going to be used for all the following power divider designs. Each design will have different values of physical length and width of the lines.

5.2.3. Simulation of a real 3 dB/0º Wilkinson at the first stage

The output ports of the power divider should have a distance between them equal to $\lambda/2$. This occurs because the array elements should be separated by half-wavelength. It is important to design the power dividers considering the layout from the beginning. The first stage is composed by four power dividers. In the figure 5.10 is shown the layout of the real Wilkinson at the first stage.

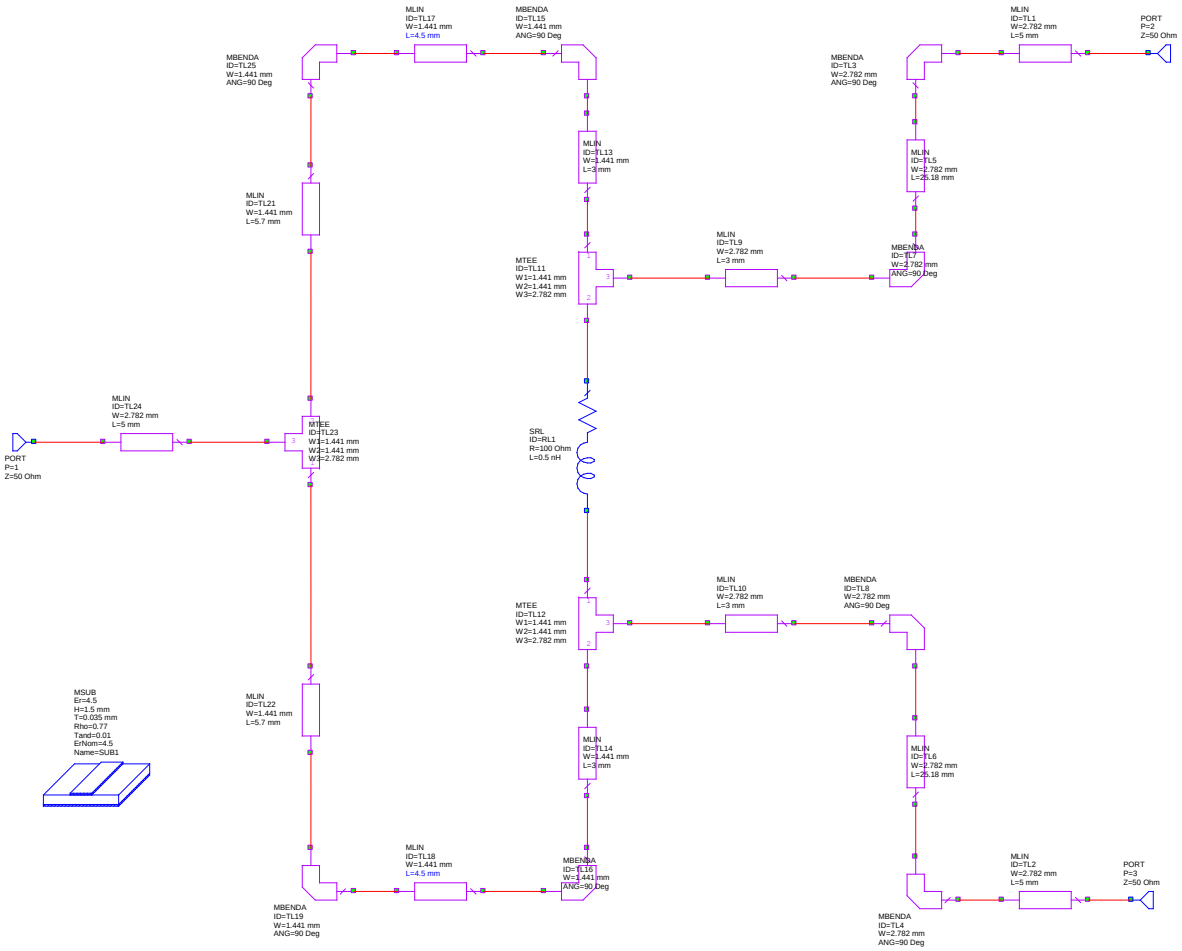


Figure 5.9. Schematic of a real 3 dB/0º Wilkinson.

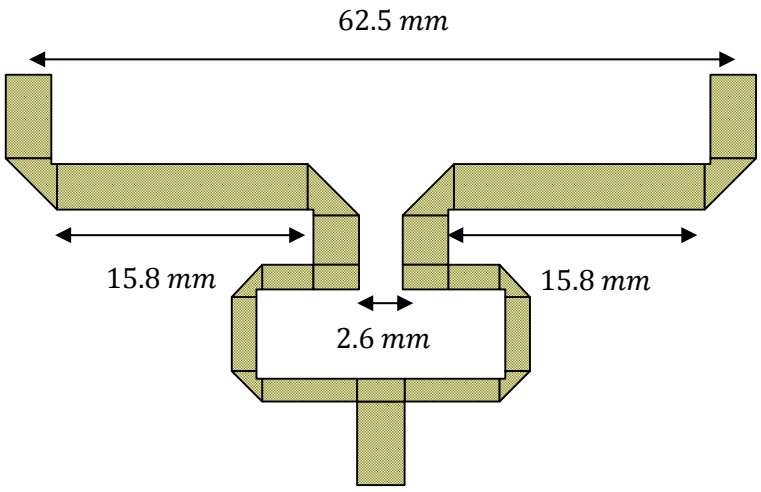


Figure 5.10. Layout of the real 3 dB/0º Wilkinson at the first stage of the feed network.

The schematic used for the above layout is the same as the figure 5.9. Once the length and the width of the lines are set by the tool TXLine, the schematic is simulated to obtain the scattering parameters. Using the Tune tool the length of the lines can be modified to optimize the frequency response of the device.

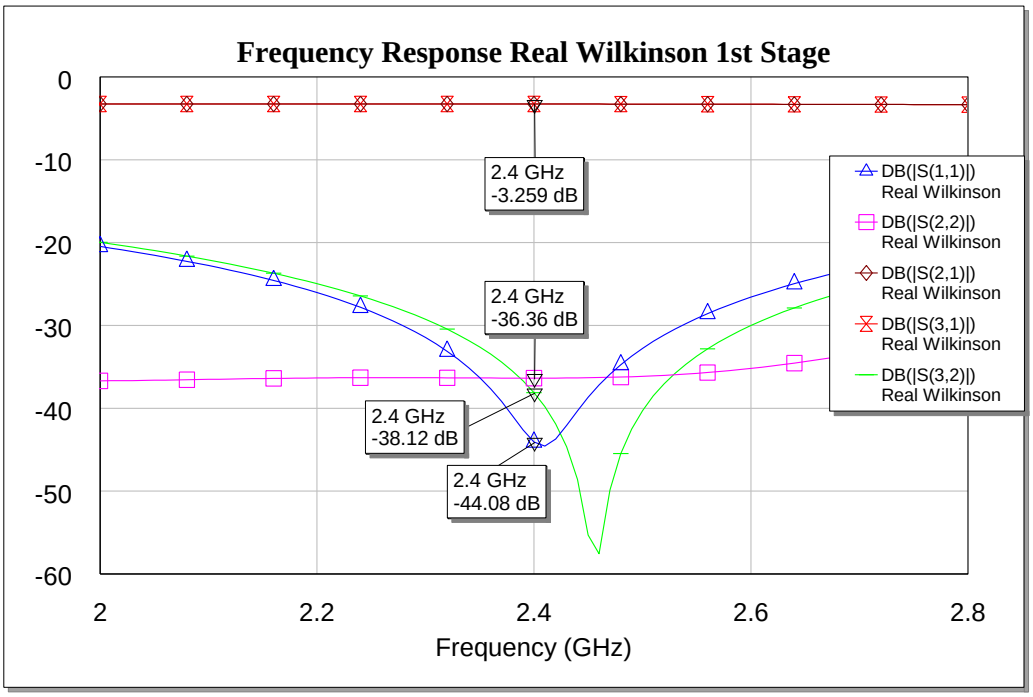


Figure 5.11. Frequency response of the real Wilkinson at the first stage.

As noted in the figure above, the value of the scattering parameters are as desired. The output ports provide half of the input power. The output ports can be considered isolated between them:

$$S_{32}(dB) = 20Log|s_{32}| = -38.03 \tag{5.2.3.1}$$

This can be expressed in linear scale as follows:

$$|s_{32}| = 10^{-\frac{38.03}{20}} = 0.0125 \simeq 0 \tag{5.2.3.2}$$

Therefore it can be considered a good approach. Also the reflection coefficient is almost zero. Then the first stage of the design of power dividers it can be considered as a good solution.

5.2.4. Simulation of a real 3 dB/0º Wilkinson at the second stage

The second stage of the feed network is composed by two power dividers. The layout needed is shown in the figure 5.12.

The length addition of the output lines of the second stage Wilkinson of the feed network is not exactly half of the distance between outputs, because it has to be considered the space where the resistance is connected. More other details have to be considered such as the width of the lines. Therefore, the schematic of the Wilkinson couplers in the second stage of the feed network is the same as the first stage, but changing the values of the lines at the output ports. Although the length of the output ports is modified, the frequency response should be the same as the first stage power dividers.

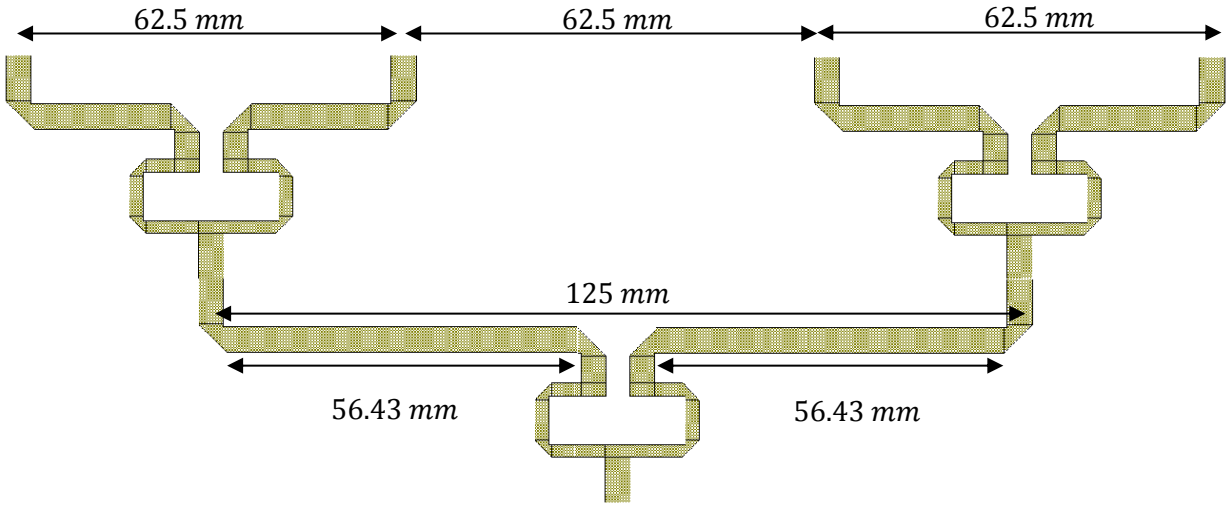


Figure 5.12. Layout of the second and first stage of the feed network.

The frequency response of the Wilkinson power dividers at the second stage is shown in the figure 5.13.

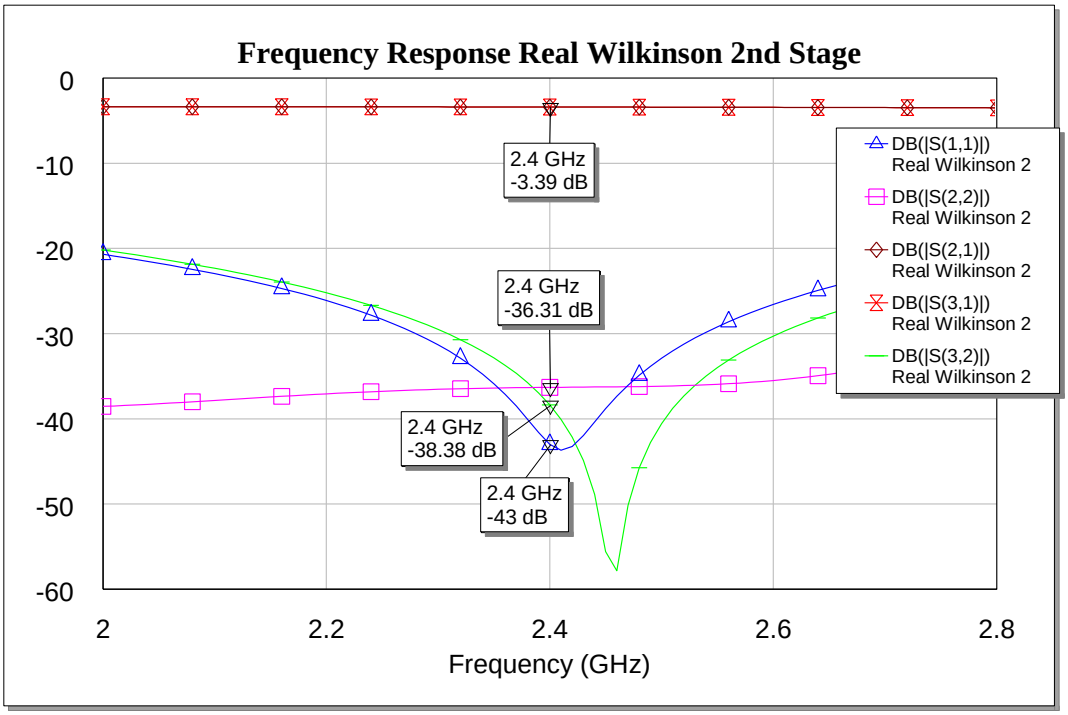


Figure 5.13. Frequency response of the real Wilkinson at the second stage.

The power reflected by the power divider is very small. In addition the output ports can be considered isolated between them because of the small value of the scattering parameter s_{32}. Finally, the output ports provide approximately half of the input power.

5.2.5. Simulation of a real 3 dB/0º Wilkinson at the third stage

The third stage of the feed network is composed by only one power divider. The layout considering all the stages is shown in the figure 5.14.

The schematic of the Wilkinson coupler in the third stage of the feed network is the same as the first stage, but changing the values of the lines at the output ports. The length of the output ports are shown in the figure 5.14. After setting the length of the output lines, the power divider has to be simulated individually. The frequency response obtained in simulation is shown in the figure 5.15. The results obtained are similar to the frequency response of the power dividers at the first and second stage. Therefore the real 3 dB/0º Wilkinson at the third stage works as desired.

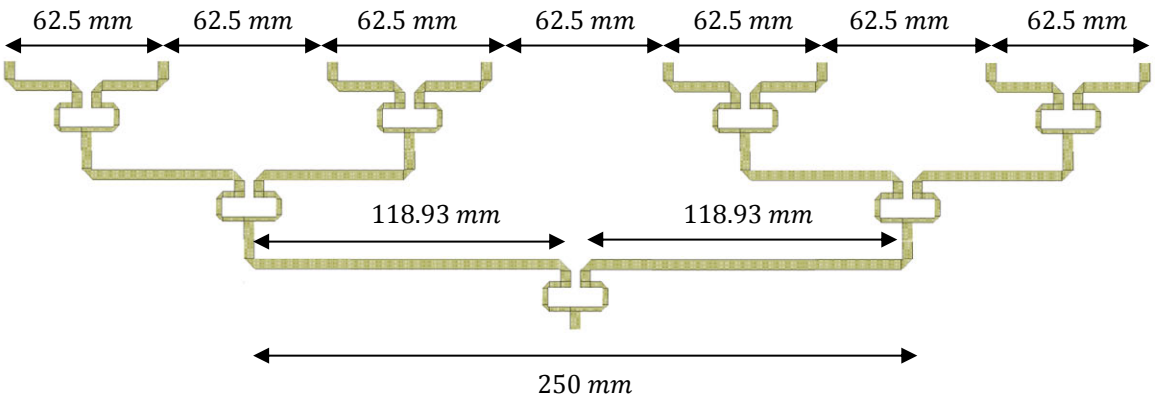


Figure 5.14. Layout of the complete feed network.

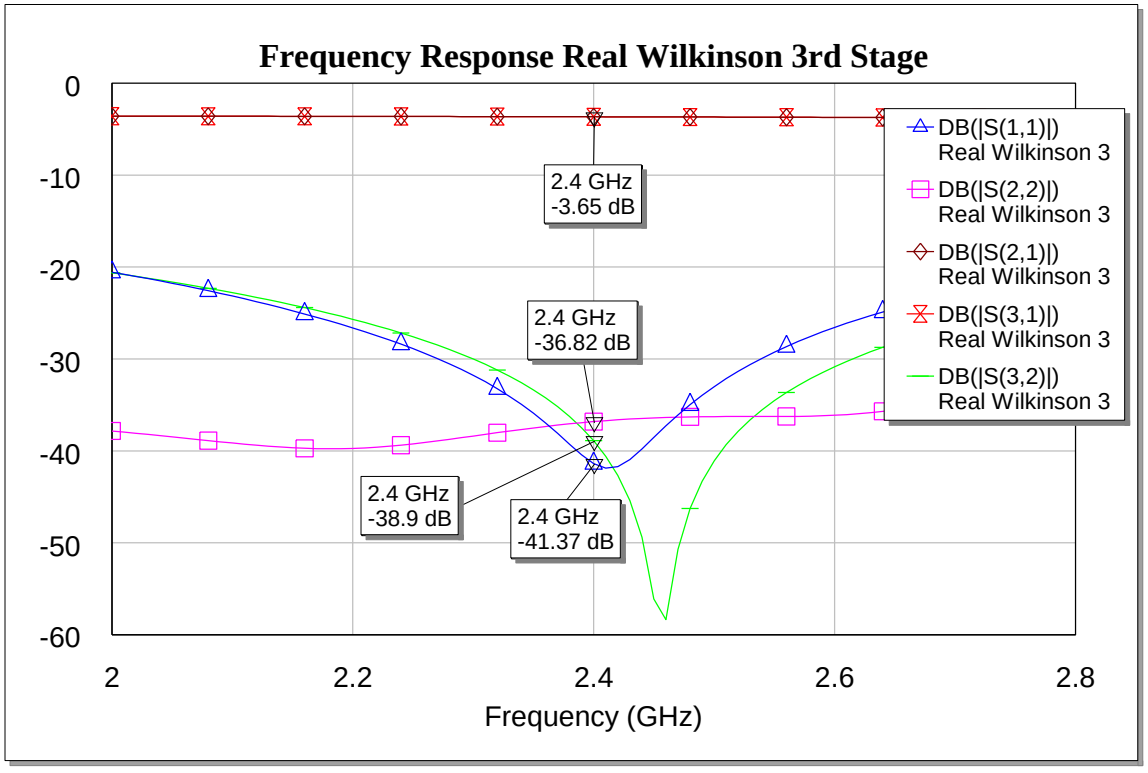


Figure 5.15. Frequency response of the real Wilkinson at the third stage.

5.3. Parallel-fed network simulation

Once the Wilkinson power dividers of each stage have been simulated and optimized individually, they should be connected forming the parallel-fed network shown in the following figure.

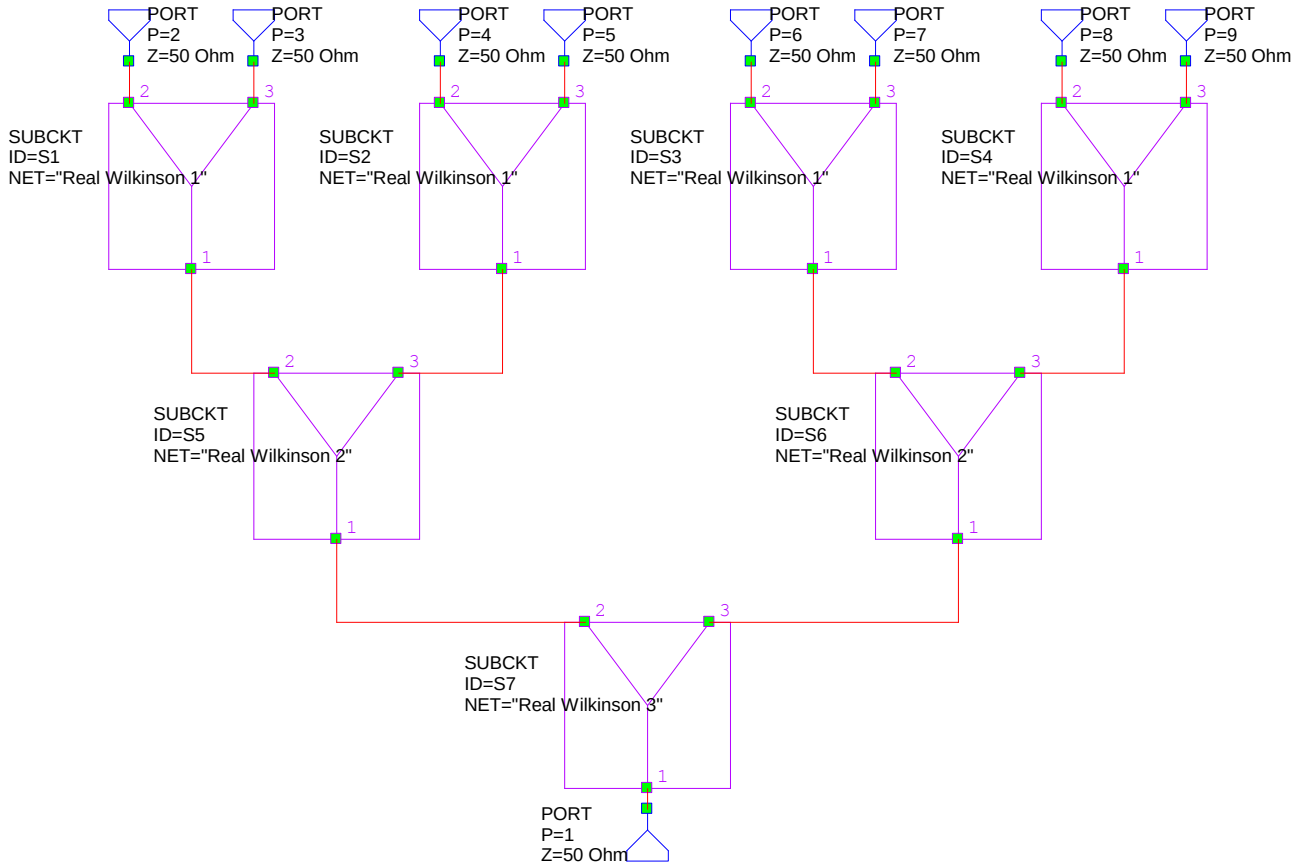


Figure 5.16. Schematic of the parallel-fed network.

The simulation will reflect the operation of each stage of the feed network. The above schematic consists of a ten-port network. This results in a large number of scattering parameters. The most important are which describe the return losses, transmission and the isolation between output ports. The frequency response is shown in the following figures.

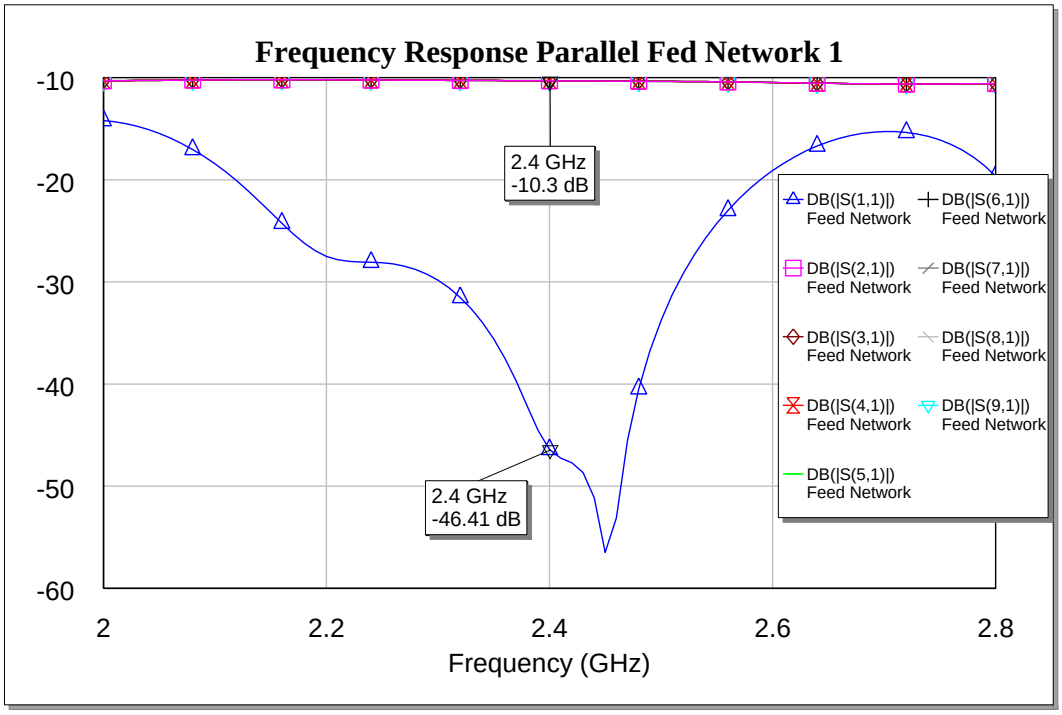


Figure 5.17. Return loss and transmission of the parallel-fed network.

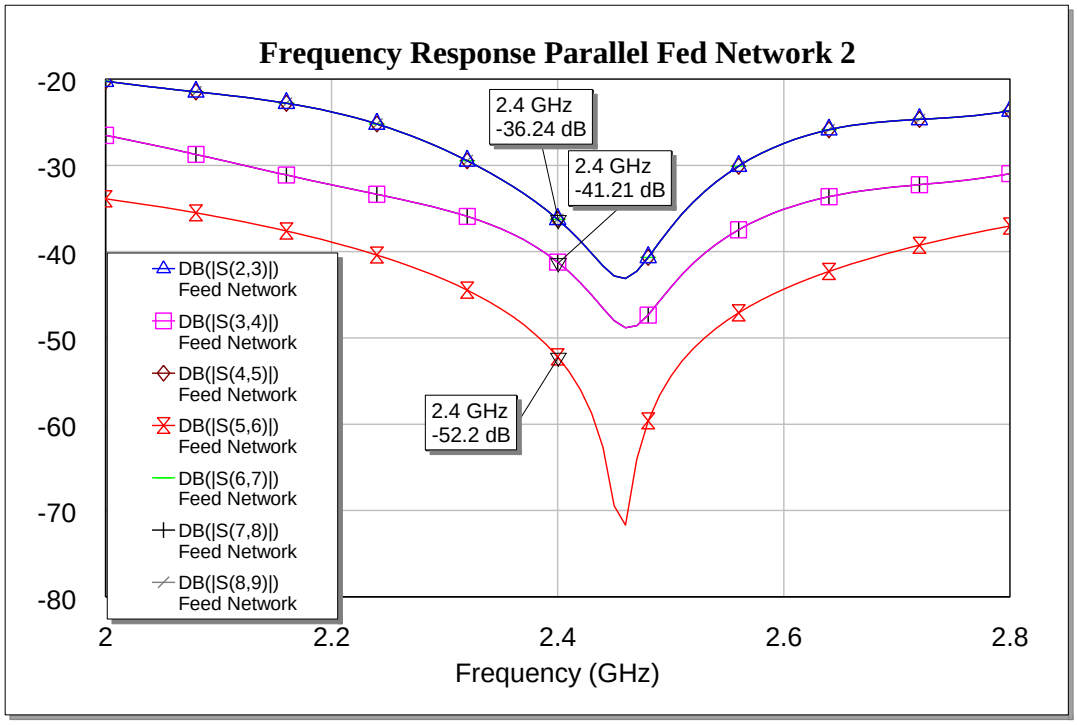


Figure 5.18. Isolation between some output ports of the parallel-fed network.

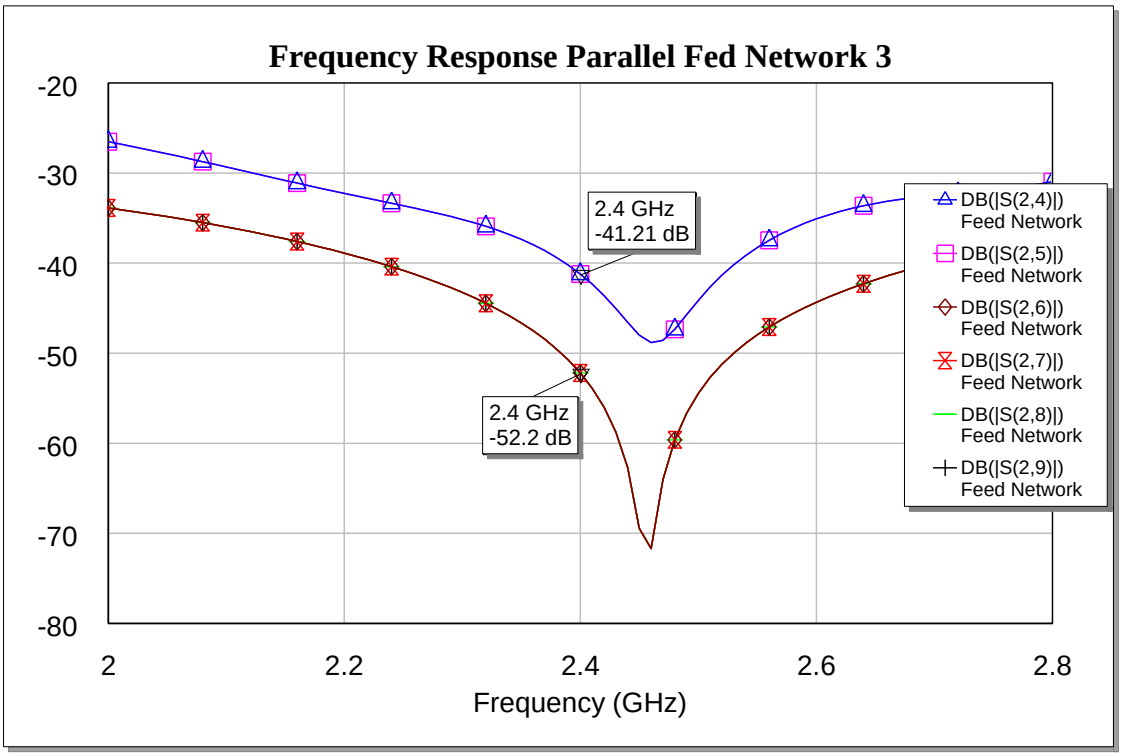


Figure 5.19. Isolation between some output ports of the parallel-fed network.

As noted the return losses of the feed network are:

$$S_{11}(dB) = 20Log|s_{11}| = -46.41\ dB \tag{5.3.1}$$

In the linear scale of values this is approximately zero:

$$|s_{11}| = 10^{-\frac{46.41}{20}} = 4.78 \cdot 10^{-3} \simeq 0 \tag{5.3.2}$$

In addition the transmission characteristics of the network can be seen by the following scattering parameters:

$$S_{21}(dB) \simeq S_{31}(dB) \simeq \cdots \simeq S_{91}(dB) \simeq -10.3 \tag{5.3.3}$$

In the linear scale this is approximately:

$$|s_{21}|^2 \simeq |s_{31}|^2 \simeq \cdots \simeq |s_{91}|^2 = 10^{-\frac{10.3}{10}} = \frac{1}{10.7} \tag{5.3.4}$$

The value expected at the output of the feed network should be:

$$|s_{21}|^2 = |s_{31}|^2 = \cdots = |s_{91}|^2 = \frac{1}{8} \tag{5.3.5}$$

By using the equations (5.1.13) and (5.1.14) the ratio of the output power to the input power is:

$$\frac{P_{OUT2}}{P_{IN1}} = \frac{P_{OUT3}}{P_{IN1}} = \cdots = \frac{P_{OUT9}}{P_{IN1}} = \frac{1}{8} \tag{5.3.6}$$

Therefore:

$$P_{OUT2} = P_{OUT3} = \cdots = P_{OUT9} = \frac{1}{8} P_{IN1} \tag{5.3.7}$$

In the simulation results the ratio of the output power to the input power is not exactly as desired, but it is a good approximation.
The isolation between output ports can be seen in the figure 5.18 and 5.19. At any case the values of the scattering parameters at the design frequency are below -30 dB, that is a good approximation. Therefore the feed network designed will operate as desired at 2.4 GHz.

6. BIAS NETWORK DESIGN

There are several solutions to perform the automatic operation of the phased array antenna. As it has been mentioned before the system will be implemented by a serial communication between a computer and a micro-controller.

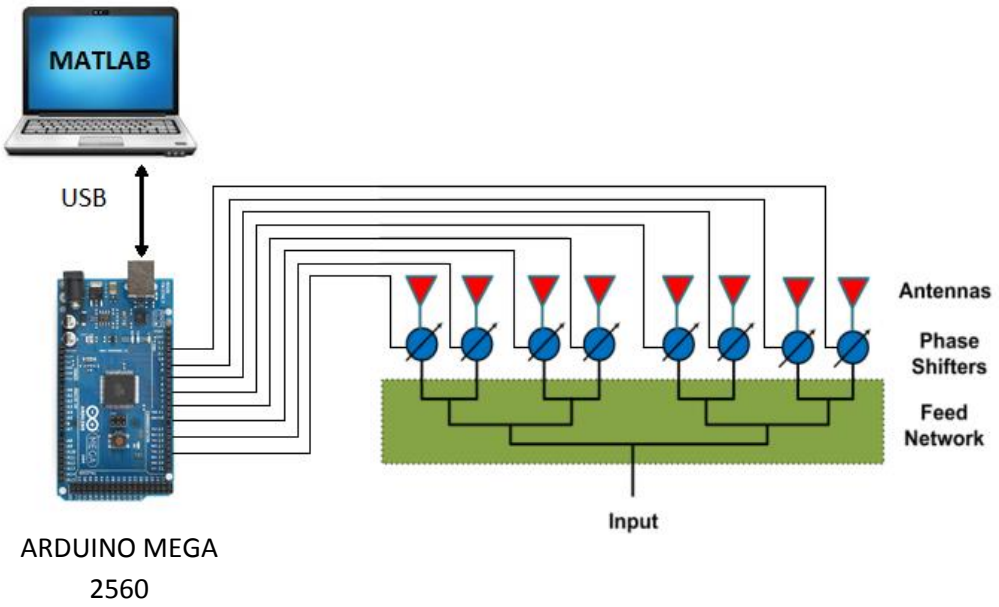


Figure 6.1. Scheme of the bias network.

The software running in the computer consists on a MATLAB function which generates several scan angles. From each value of the scan angle, it is obtained the phase shift between two consecutive radiating elements. Once the phase difference is known, the software calculates the phase it has to be applied to the signal on each radiating element. As mentioned before, the phase shifters apply a phase into the incoming signal depending on a control DC voltage. Therefore based on the relation between control voltage applied and phase shift, the pc software could obtain the corresponding voltages it should be applied at each phase shifter. Then the DC voltages values are sent from a laptop by MATLAB to the micro-controller Arduino Mega, which is responsible for generating the control voltages on its analog output ports. It has been chosen Arduino Mega 2560, because of several reasons:

- Open-source electronics prototyping platform based on flexible, easy-to-use hardware and software.
- Programming language simple.
- Supports a serial communication USB.
- Contains 15 analog PWM output ports (2-13 and 44-46), which could provide DC voltages from 0 to 5 V.

The solution proposed above, implies that different functionalities should be developed by MATLAB software and other functionalities are developed by Arduino. These two systems depend on each other. Both parts need to establish a reliable communication to send and receive data. In the following figure the architecture of the software needed to implement the bias network design is showed.

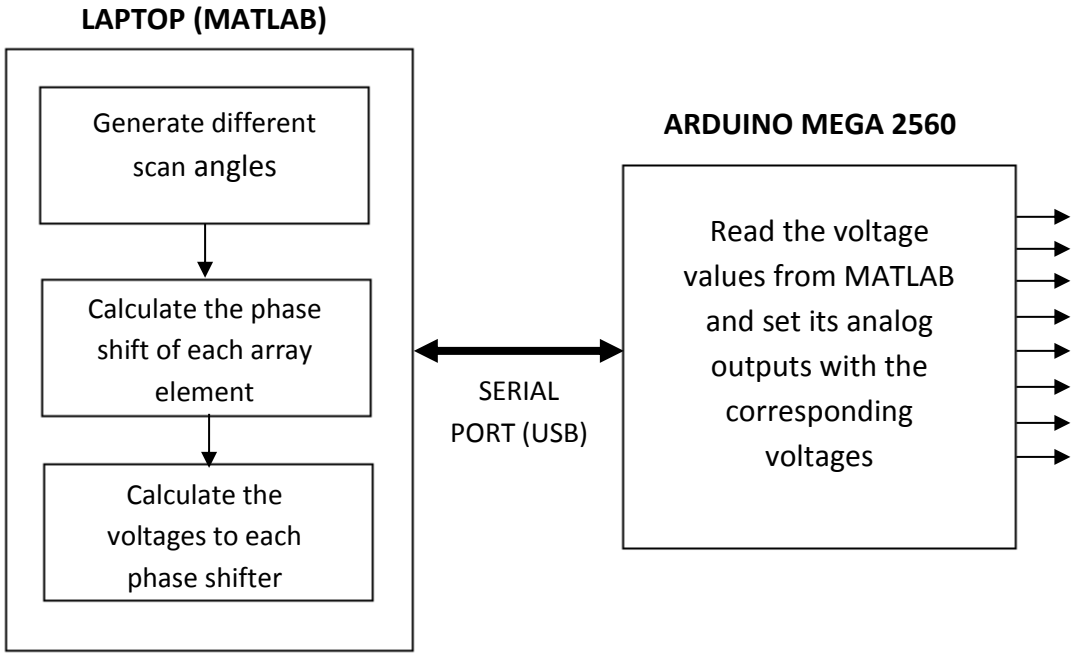


Figure 6.2. Software architecture of the bias network.

As noted, the solution will consist of several independent programs with different programming languages. They will be connected and they will be able to exchange information by a serial port (USB). There are two possible solutions to implement the above system:

- The control voltage values can be calculated from the host computer program developed by MATLAB and sent to the Arduino. Then the micro-controller processes these values and set the control voltages to its analog output ports using a server program developed for this particular functionality, receiving bytes that represents the control voltages.

- Using MATLAB Support Package for Arduino, developed by MathWorks. This software allows communicating with Arduino over a serial port. It consists on a MATLAB API in the host computer and a server program that runs on the Arduino. So far, this solution has the same architecture as the first one. The main difference between them is that, using the MATLAB Support Package for Arduino, the host computer program defines several functions. These functions are able to control the Arduino from the host computer,

without any knowledge of programming language in Arduino environment. Hence, the Arduino can be treated as a black box executing orders from the host computer.

6.1. Phase shifters

The model of phase shifter used in this design is an HMC928LP5E a 450º analog phase shifter working in the 2-4 GHz band. This device is controlled via an analog control voltage from 0 to 13 volts. It provides different phase shifts of 0 to 450 degrees. Besides it is characterized to have very low insertion losses and low phase error, which makes this type of phase shifters to be ideal for radar applications.

The relationship between the control voltage and the phase shift is given in the next graph provided by the datasheet.

- HMC928LP5E Characteristics:
  - Octave bandwidth: 2-4 GHz
  - Up to 450º phase shift
  - Low insertion loss: 3.5 dB
  - Low phase error: ±5

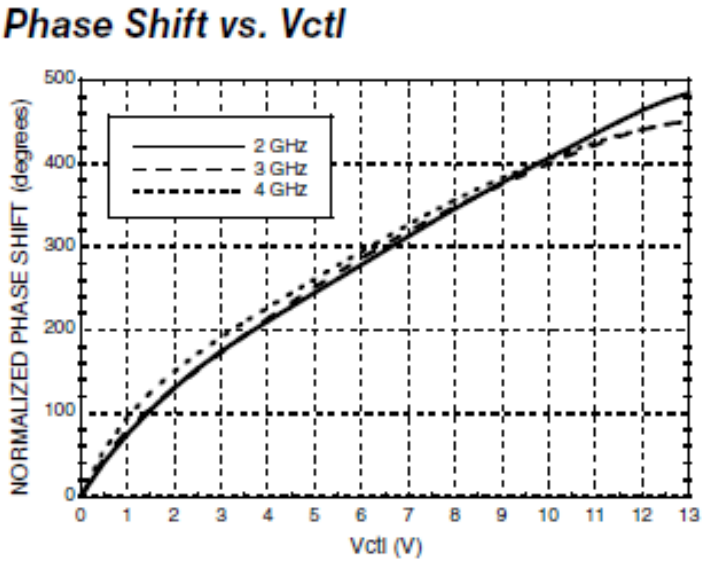


Figure 6.3. Phase shift vs. control voltage of the phase shifters.

This gives an approximation of the voltage needed to obtain certain values of the phase shift. Phase shifts about 360 degrees can be obtained with a voltage around 8.5-9 V. Therefore to ensure a well-operation of the system the maximum value of DC voltage provided to the phase shifters will be 10V corresponding to 400 degrees phase shift. Since Arduino can just provide DC voltages from 0 to 5, several operational amplifiers will be used to amplify the voltages provided by the micro-controller.

As a result of the architecture of the system, the phase shift of each array element is calculated to generate a determined scan angle. In order to obtain the phase shift, the control voltage of each phase shifter has to be a determined value given by the above graph. The relationship between the control voltage and the phase shift can be obtained more accurately by taking measurements. But it is possible to obtain a first approximation with a function in MATLAB called graph_picker.m. This function allows obtaining the values of each point in a graph imported as image. If this function is called in the command window a new window is opened. In this new window the image with the graph information should be load. After that the calibration point must be defined. The calibration points consist on the minimum and maximum values of the x and y axis. Finally the option 'points' is used to put several markers in the point we want to obtain the information (coordinates). The points selected appear in red color. The greater the number of points chosen the more accurate the value of the corresponding voltage at a given value of phase shift. The following figure shows the procedure mentioned before.

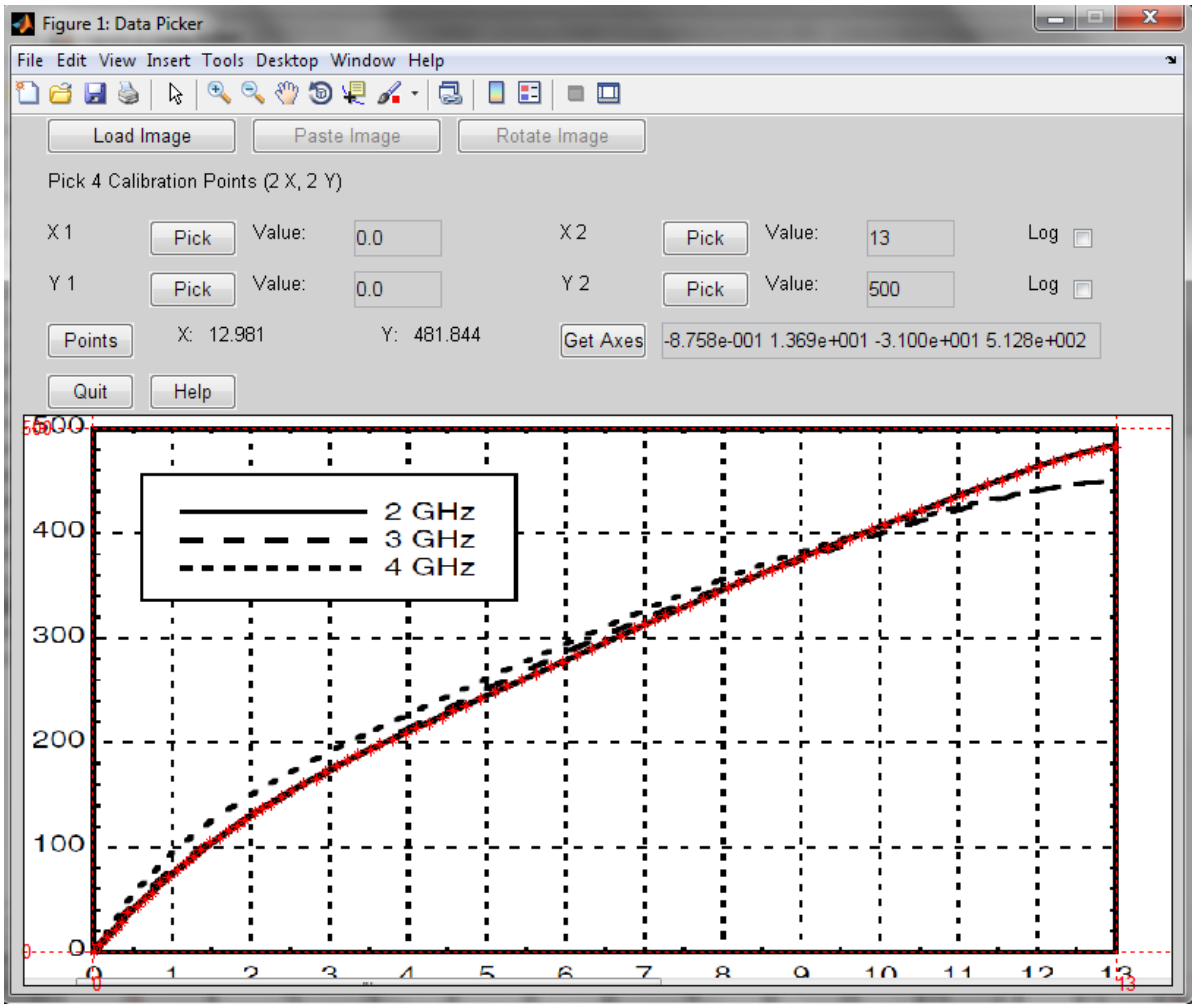


Figure 6.4. Points of the phase shift vs. control voltage graph, set by the graph_picker function.

After setting the points in the graph, the function returns the coordinates with all the points selected in the image. These coordinates will be used in the next section in order to obtain the control voltages needed to set a determined phase shift in the phase shifter. The number of point chosen is 96.

6.2. MATLAB Software

This part of the bias network system has to cover several functionalities:

- Generate several scan angles to change the directivity of the array antenna. This can be implemented with a function generating several scan angles randomly or introducing the desired scan angle by the keyboard.
- Calculate the phase shift between two consecutive array elements that results in the corresponding scan angle.
- Calculate the phase applied on each array element. These values depend on the number of array elements. Although the array antenna used has 8 array elements the software should work with different values of array elements (open system).
- Calculate the DC voltages should be applied to each phase shifter to obtain the corresponding phase shift on each array element.
- Send all the DC values to the Arduino Mega 2560. A serial communication between MATLAB and Arduino is required.

Each functionality will be implemented by a simple MATLAB function.

6.2.1. Function generate_scan

This function is responsible for generating different values of scan angles. There is no input parameter needed. Each time the function is called it returns a scan angle. The scan angle is a random variable with uniform distribution from -90 to 90 degrees. Its syntax is as follows:

$$[scan\_ang] = generate\_scan()$$

Where:

Symbol	Description	Units	Status
scan_ang	Scan angle	Degrees	Output

Table 6.1. Parameters of the function generate_scan.

The code of this function is shown in the appendix B (B.1).
The scan angle is defined as the angle with respect the direction of the signal if there is any phase applied to each element array.

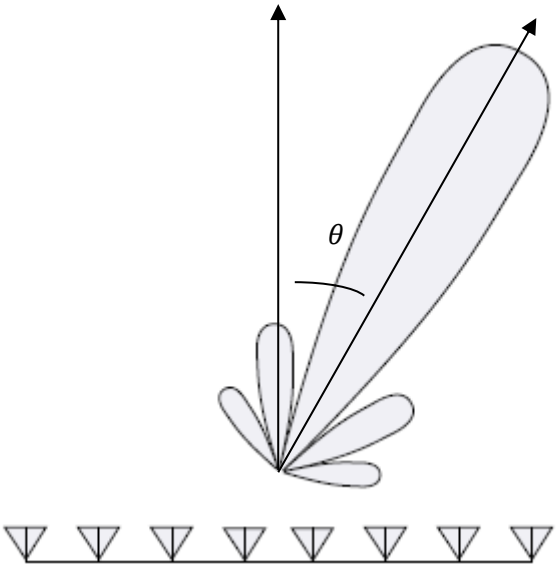


Figure 6.5. Generic scan angle of the phased array.

The values of the scan angle are positives from the vertical to the right and negatives from the vertical to the left.

6.2.2. Function calculate_phaseShift

This function calculates and returns as output parameter the phase shift, between two consecutive array elements, that generates the corresponding scan angle introduced as input parameter. Its syntax is as follows:

$$[phase\_shift] = calculate\_phaseShift(freq, d, scan\_ang)$$

Where:

Symbol	Description	Units	Status
phase_shift	Phase shift	Degrees	Output
freq	Frequency	Hz	Input
d	Distance between array elements	meters	Input
scan_ang	Scan angle	Degrees	Input

Table 6.2. Parameters of the function calculate_phasesf.

The phase shift between two consecutive array elements is obtained considering the receiving case of the antenna. A scheme of the array antenna with the beams at a given scan angle is showed in the following figure:

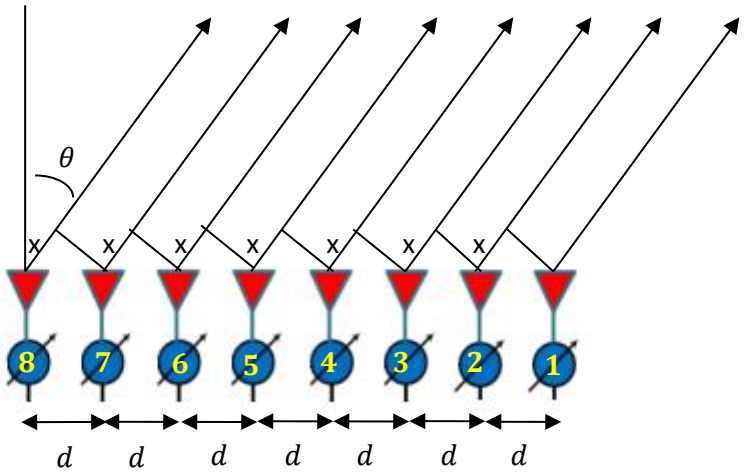


Figure 6.6. Scheme of the phased array forming a beam.

It can be noted that the received signal experiences a phase difference in the element 2 with respect to the element 1. The distance difference is denoted by the following equation:

$$x = dsin\theta \tag{5.2.2.1}$$

The phase difference of the signal with respect to the previous can be expressed as:

$$\Delta\alpha = \beta x = \frac{2\pi}{\lambda}x \tag{5.2.2.2}$$

Therefore the phase shift can be expressed as function of the scan angle by the following equation:

$$\Delta\alpha = \frac{2\pi dsin\theta}{\lambda} \tag{5.2.2.3}$$

These three equations are used in the above mentioned function to calculate the phase shift between two consecutive elements. The scan angle can be positive or negative, but in this function only the absolute value of the scan angle is considered. The code of this function is shown in the appendix B (B.2).

6.2.3. Function calculate_phases

This function calculates the phases of each array element considering all the elements equidistant. Here it is used the phase shift between two consecutive array elements, previously obtained, as an input parameter. It returns a matrix with dimensions 1xN

where N is the number of radiating elements which compose the array antenna. The number of elements it is introduced as input parameter as well. Its syntax is as follows:

$$[phases] = calculate\_phases(N, phase\_shift)$$

Where:

Symbol	Description	Units	Status
phases	Phases of each array element	Degrees	Output
N	Number of array elements	None	Input
phase_shift	Phase shift between two array elements	Degrees	Input

Table 6.3. Parameters of the function calculate_phases.

This function consists of a loop calculating the phase of each array element from the first element to the last. The phase of the first array element is zero, this value is incremented the value of the phase shift and set to the next array element. The procedure is the same for the other array elements. The code of this function is shown in the appendix B (B.3).

6.2.4. Function toggle

This function is used to change the order of the elements in a vector. The last element becomes the first and vice versa. The output parameter returns the vector with the same values but inverted order. Its syntax is as follow:

$$[vector2] = toggle(vector1)$$

Where:

Symbol	Description	Units	Status
vector1	Vector which contains the values which order is going to be inverted	Degrees	Input
vector2	Vector with inverted order of elements	Degrees	Output

Table 6.4. Parameters of the function toggle.

The function toggle.m is used when the scan angle desired is negative. If the scan angle is negative, the function calculate_phasesf.m returns a negative phase shift. The magnitude of this phase shift would be the same as the phase shift corresponding with the same scan angle but with positive sign. It can be seen with an example.

Considering the following characteristics:

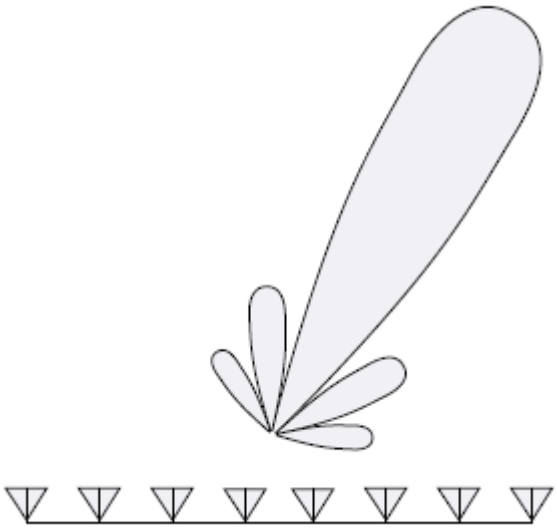
- Frequency: $f = 2.4\ GHz$
- Number of array elements: $N = 8$
- Spacing between elements: $d = {}^{\lambda}/_{2}$

There are two possible results depending on the sign of the phase shift:

CASE 1	CASE 2
Scan angle: 45º	Scan Angle: -45º
Phase shift: 127.28º	Phase shift: -127.28º

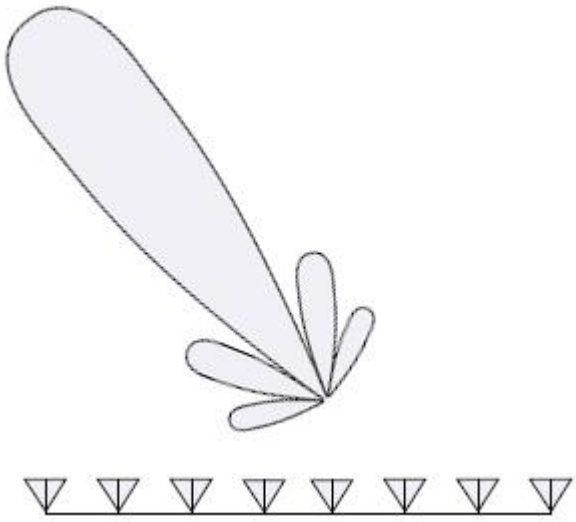
Table 6.5. Cases with different scan angle and phase shift.

If the first array element has a phase shift equal to zero. A positive phase shift means that the element following the first array element has 127.28 degrees of phase shift with respect to the first. Therefore the phase shift of each array element would be as depicted in the figure 6.7. On the other hand a negative phase shift means that the element following the first one has phase shift of -127.28º degrees with respect to the first array element. In other words the first array element has a phase shift of 127.28º degrees with the following. Therefore to obtain a scan angle of -45º, the phases of each array element could be as shown in the figure 6.8. For that reason when the scan angle is negative, the phase shift and the phases of each element are calculated considering the scan angle positive. After calculating the phases of each element array the function toggle is called. The function inverts the order of the vector containing the phase of each array element. Therefore once the voltages of each phase shifter are obtained and sent to the micro-controller, the resulting scan angle is negative. The code of this function is shown in the appendix B (B.4).



8	7	6	5	4	3	2	1
170.96º	43.68º	276.4º	149.12º	21.84º	254.56º	127.28º	0º

Figure 6.7. Scan angle of 45 degrees and phases of each radiating element.



8	7	6	5	4	3	2	1
0º	127.28º	254.56º	21.84º	149.12º	276.4º	43.68º	170.96º

Figure 6.8. Scan angle of -45 degrees and phases of each radiating element.

6.2.5. Function get_voltages

This function calculates the voltage should be applied to the phase shifter in order to generate the desired phase shift introduced by input parameter. The input parameter is a matrix of 1xN elements, where each element is the phase shift of each array element and N is the number of radiating elements. The output is a matrix of 1xN as well with the resulting value of voltages. Its syntax is as follows:

$$[voltages] = get\_voltages(phases)$$

Where:

Symbol	Description	Units	Status
phases	Matrix 1xN. Phase shift provided by each phase shifter	Degrees	Input
voltages	Matrix 1xN. Control voltage of each phase shifter	Volts	Output

Table 6.6. Parameters of the function get_voltages.

In this function the points of the graph control voltage versus phase shift obtained by the function graph_picker.m are used to obtain the corresponding values of voltages. The number of points used is 92. Therefore in order to obtain the voltages, corresponding to points which have not been chosen in the function, it should be used interpolation between known points. The syntax of the interpolation function used is as follows:

$$X1 = interp1(Y, X, Y1, 'spline')$$

Where the parameters are defined in the table 6.7.

Therefore the function get_voltages just has to call the function interp1 within a loop in order to obtain all the voltage values corresponding with different phase shifts. The code of this function is shown in the appendix B (B.5).

Symbol	Description	Units	Status
X	Vector with the x coordinates at some points of the graph chosen	Volts	Input
Y	Vector with the y coordinates at some points of the graph chosen	Degrees	Input
X1	x coordinate of the point desired	Volts	Output
Y1	y coordinate of the point desired	Degrees	Input
'spline'	Sets a piecewise cubic interpolation	None	Input

Table 6.7. Parameters of the function interp1.

6.2.6. Function map

The phase shifters used in this project need DC control voltages from 0 to 10 V. After calculating the control voltages should be sent to each phase shifter, they need to be mapped. This function coverts the amount, within the interval [fromLow,fromHigh], introduced as input parameter. The output value represents the same amount as the input value but within the interval [toLow,toHigh]. Its syntax is as follows:

$$[num] = map(val, fromLow, fromHigh, toLow, toHigh)$$

Where the parameters are defined in the table 6.8.

The function get_voltages.m provides voltages with several decimal numbers. These values have to be sent through the serial port. A serial communication allows sending strings and binary data. As mentioned before the Arduino Mega 2560 can provide DC voltages through its analog ports, from 0 to 5 V. In order to set the voltages at the output ports it is important to note that analog ports only understand values from 0 to 255. For example if an output port is set with value 255, the value in volts is 5. In addition the Arduino sends binary data byte to byte. One byte consists of 8 bits, which represent unsigned integer numbers from 0 to 255. For this reason a good solution would be converting the control voltage values desired within the interval [0,10] into values within the interval [0,255]. This can be performed by calling map function with the following input values:

$$[num] = map(val, 0{,}10{,}0{,}255)$$

Where variable 'val' represents each value of the output vector as a result of calling the function get_voltages. As noted the micro-controller provides voltages up to 5 V, but in the function map.m the values converted can be up to 10 V. This is due to the presence of a set of operational amplifiers at the output of the Arduino analog ports. If the desired voltage at one output is 10 V, the microcontroller sets this output with 255, which corresponds with 5 V. Hence, after the amplifiers the final voltage would correspond with 10 V, as desired. This is a good approximation to set the voltage values at the output ports. The code of this function is shown in the appendix B (B.6).

Symbol	Description	Units	Status
val	Value within the range [fromLow,fromHigh], which is going to be converted	Volts	Input
fromLow	Minimum value of the original scale	Volts	Input
fromHigh	Maximum value of the original scale	Volts	Input
toLow	Minimum value of the destination scale	None	Input
toHigh	Maximum value of the destination scale	None	Input
num	Resulting value, within the range [toLow,toHigh] after conversion	None	Output

Table 6.8. Parameters of the function map.

6.3. Arduino Mega 2560 Environment and Software

The micro-controller Arduino Mega 2560 on the board is programmed using the Arduino programming language that allows defining variables, structures and functions needed to implement the bias network design. In order to develop the functions and the software running in the micro-controller it is used an environment. The environment is written in Java and based on processing, avr-gcc, and other open source software. The version used here is Arduino 1.0.5 which is compatible with Arduino Mega 2560.

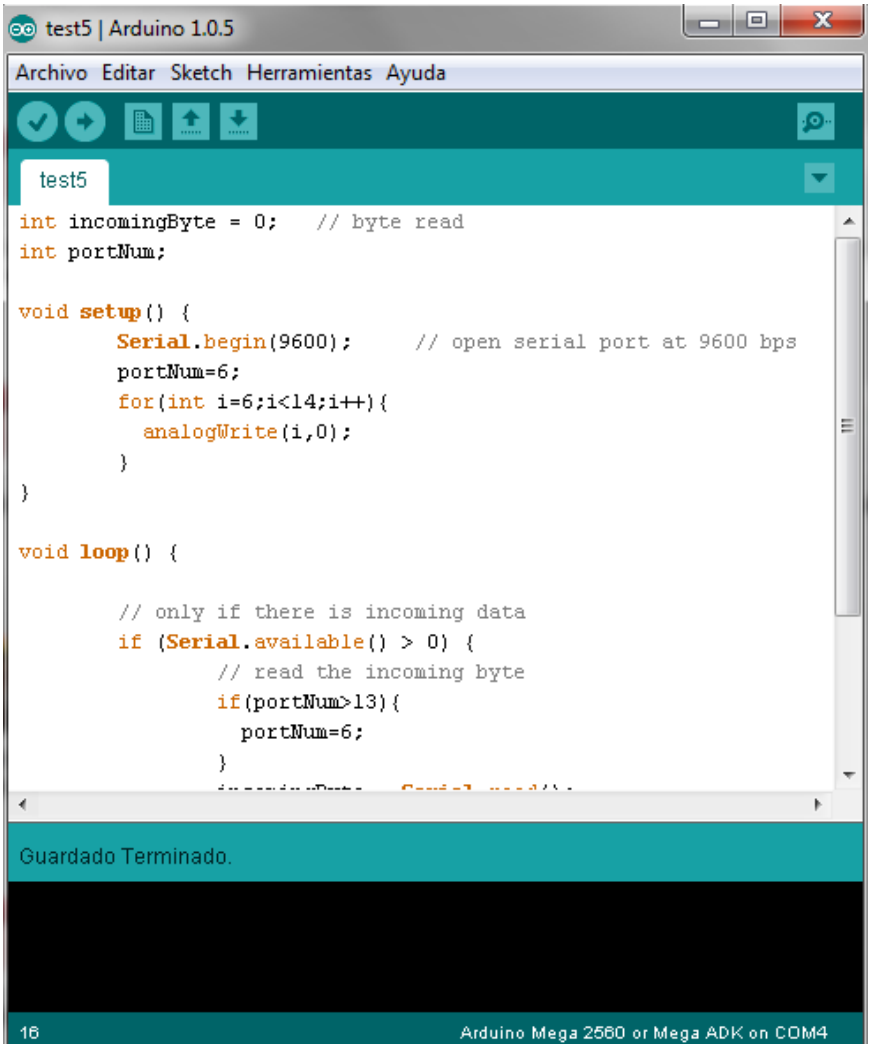


Figure 6.9. Arduino environment.

In the figure above it can be seen the environment for Arduino. The environment is used to write and compile the code it is going to be run by the micro-controller. As it can be seen on the bottom it is configured to Arduino Mega 2560 micro-controller. After compiling, the programs are sent to the Arduino board by the port 'COM4' (USB).

All the programs in Arduino have the same structure:

- Setup function: This function is run once. Here are usually defined or initialized variables used in the loop function, such as serial communication. Besides can be defined the status of some digital or analog port as input or output.

- Loop function: The loop function is running constantly. Here are defined all the instructions which need to be executed always. As an example, we can think of a serial communication receiving values from a computer. The software has to check without stop if there is some data to read.

- Other functions: Additional functions can be codified to simplify the program and make it more understandable. These functions can be used in both setup and loop functions.

Once the environment and the structure of a simple program in Arduino have been defined, the server program running in Arduino can be implemented. So far the system that implements the bias network generates a scan angle which the antenna has to be set. Then several MATLAB functions are responsible for calculating the phase shift of each array element of the antenna. Finally, the control voltages, which have to be applied to each phase shifter in order to get the corresponding phase shift, are calculated and sent to the micro-controller. From now on it is used Arduino software to process the received data and set the analog output ports with the desired control voltages.

As mentioned before the following part of the system can be implemented using two different solutions:

- Program that receives bytes representing the control voltage values and sets the analog output ports with the corresponding values received.

- MATLAB Support Package for Arduino, using the functions available, in the host computer.

In the next sections these two solutions are going to be developed, analyzing the advantages of both of them.

6.3.1. Arduino software

The first solution consists on a simple program loaded in the micro-controller. This program is responsible for receiving bytes, representing the voltage values from the serial port and setting the output analog ports with the corresponding control voltage, as mentioned before. Considering the above mentioned structure of a program in Arduino, each function and its functionality are going to be detailed next. The name of the program is set_voltages.pde. The code of the program can be seen in the appendix C (C1).

- Function Setup: A serial communication is required to receive the bytes from MATLAB. Therefore the setup function will define a serial communication by the serial port ‘COM4’ (USB), with a baud rate of 9600.

```
Serial.begin(9600);
```

The output voltages are supplied by the analog ports of the micro-controller. Then the analog ports have to be configured as output ports. The syntax to set a port as output is as follows:

```
pinMode(port,OUTPUT);
```

Where 'port' is the number of the analog port it is going to be configured. The number of ports configured is eight. This is because the array antenna has eight radiating elements and each one requires a phase given by the phase shifter. The micro-controller Arduino Mega 2560 has fifteen analog ports (PWM) from 2 to 13 and from 44 to 46. The ports chosen in this case are from 2 to 9. In addition the output ports are initialized to zero volts. This is made to establish a reference, because the initial output values are not known. The instruction that assigns the voltage to an analog output port is the following:

```
analogWrite(portNum,data);
```

Where 'portNum' is the number of the port in the board and 'data' is the voltage which is going to be set the analog output port. The variable 'data' takes values from 0 to 255. Several output ports have to be initialized, hence the 'analogWrite' instruction is used within a 'for' loop.

- Function loop: The function loop receives constantly bytes, one by one, which represents the control voltage values divided by two. These values go from 0 to 255. The instruction used to read bytes from the serial port is the following:

```
incomingByte = Serial.read();
```

The analog ports are set with the values read, using the instruction 'analogWrite' mentioned before. Only eight ports are going to be used. Therefore when the Arduino receives more incoming bytes and all the analog ports are set, it has to rewrite the analog ports.
It is important to establish a little delay between two consecutive readings from the serial port, in order to avoid synchronization problems reading data. For example in the program used the delay between two consecutive readings is 100 μs.

Once the Arduino is willing to accept incoming data through the serial port, the data is sent from the host computer. In order to send the data from the computer, several scripts are going to be programmed. In each script all the voltage vales are going to be obtained, using the functions defined in the section 6.2, MATLAB software. Then the values will be mapped and sent to the micro-controller through the serial port. The main difference between them is how to obtain the scan angle desired:

- software1.m: In this script the scan angle is introduced by the keyboard.

- software2.m: Unlike the script mentioned before, here the scan angle is generated randomly. This can be possible using the function generate_scan. This function generates a scan angle using a uniform random distribution function, from -90 to 90 degrees.

The values representing the voltages needed are sent through the serial port as unsigned integers from 0 to 255. Hence, each value is represented as a simple byte (8 bits). In order to send the values through the serial port, first of all a serial communication has to be created:

```
serialPort=serial('COM4','BaudRate',9600);
```

The input parameters are the COM port, and the baud rate of the serial port. It has to match with the baud rate defined in the Arduino software. The function returns an identifier that is stored in the variable 'serialPort'. After that the port has to be opened to start sending data:

```
fopen(serialPort);
```

Then the data can be sent using the following function:

```
fwrite(serialPort,val,'uint8');
```

Where the input parameters are: serial port identifier, the value to be sent and the precision. As the values to be sent are integers from 0 to 255, they can be represented by one byte. Therefore the precision used will be unsigned integers of 8 bits, 'uint8'. Once all values have been sent the serial port has to be closed. This can be made by using the following function:

```
fclose(serialPort);
```

The scripts use the functions defined in the section 6.2 to obtain the voltage values needed to obtain the desired scan angle. Then using the above mentioned functions, the values needed to set the voltages at the output ports of Arduino are sent through the serial port. The code of both scripts is shown in the appendix C (C2 and C3).

The solution purposed above consists of a simple program in Arduino. Therefore, the execution time is short. This can be an important advantage for tracking radars with targets in motion. In this case the system has to able to change the scan angle as fast as possible.

6.3.2. MATLAB Support Package for Arduino

As mentioned before, another way to implement the bias network design is using the MATLAB Support Package for Arduino. This consists of a MATLAB program and the server program in Arduino working together. Before using the functions defined by the software in MATLAB, the package has to be installed in the host computer. The package contains the following files:

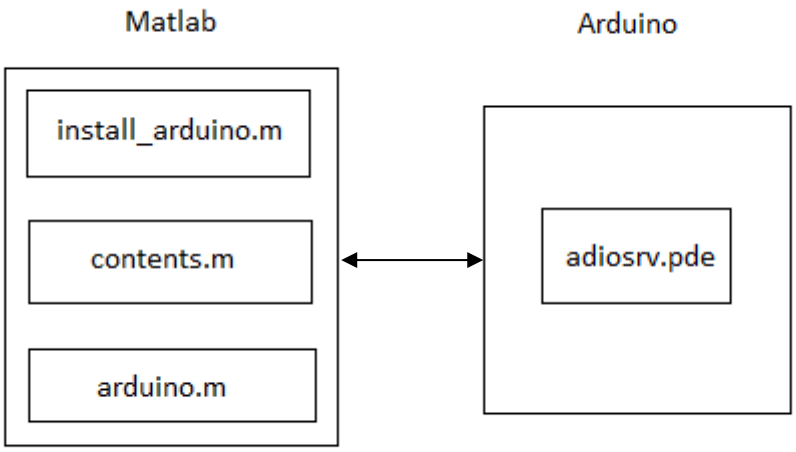


Figure 6.10. Arquitecture of the software using MATLAB Support Package for Arduino.

- install_arduino.m: Intalls the MATLAB Support Package for Arduino.
- contents.m: It is not a program file. It consist of an explanation about a new class defined in the file arduino.m, that allows performing analog/digital input and output with the Arduino Board from the MATLAB command line.
- arduino.m: Defines an arduino object and several functions that allow controlling the Arduino Board.

The package allows establishing a serial communication between MATLAB and Arduino. The file arduino.m developed in MATLAB, defines several functions which can be used to control the micro-controller from the host computer. Hence, by simply knowing how to use these functions, the micro-controller can receive the orders by the serial communication port. There is no need to know how to program in the Arduino environment. In this case the micro-controller is treated as a black box. Some of the functions used in this project are listed below.

- Connect: Creates an Arduino object, which consists in an open a connection using a serial port. It uses the right COM port as a string input argument, in this particular case 'COM4'.

```
a=arduino('COM4');
```

- Assign Pin Mode (Input/Output): Sets or gets the mode of a specified pin. Where the input parameters are the number of pin and the mode as a string ('input'/'output'). If both 'pin' and 'str' parameters are set, the function sets the pin mode. Instead if only the first parameter is set, the function gets the pin mode.

```
a.pinMode(pin,str);
```

- Analog Write: Sets the value 'val' into the analog pin 'pin'.

```
a.analogWrite(pin,val);
```

- Disconnect: Disconnect the MATLAB session from the Arduino board. This renders the serial port available for other sessions or the IDE environment.

```
delete(a);
```

After defining some of the available functions in Matlab Support Package for Arduino, then they are used in some scripts. As in the first solution, there will be two different scripts, where the voltage values are calculated to obtain the desired scan angle. In the script software3.m, the scan angle is going to be introduced by the keyboard, while in the script software4.m the scan angle is going to be a random variable with uniform distribution function from -90 to 90 degrees. The difference between the scripts in this solution and the scripts in the section 6.3.1 is that the micro-controller is controlled by using the functions defined by MATLAB Support Package for Arduino. Therefore the micro-controller analog ports are set in the host computer. The code of the scripts mentioned before is shown in the appendix C (C4 and C5).

Using the MATLAB Support Package is better than the first solution because of the reliability of the serial communication offered. It avoids any synchronization problems which are present using graphical user interfaces (GUI) with the Arduino software in the first solution set_voltages.pde. On the other hand, as this solution uses functions defined in different files and the programs are more complex, the execution time is longer than the first solution. This reduces the speed of sweep in tracking radars.

In this project the code of the functions is not going to be shown. The MATLAB Support Package for Arduino can be obtained at the official page of MathWorks.

Before testing the system is important to remember that the micro-controller Arduino Mega 2560 provides voltages values from 0 to 5 V. The phase shifters use voltages from 0 to 10 V. Therefore, a network of amplifiers is needed before providing the DC control voltages to the phase shifters.

6.4. Non-Inverted Amplifiers

As the micro-controller Arduino Mega 2560 can provide DC voltages up to 5 V and the phase shifters need 10 V at least to obtain a phase shift of 360 degrees, a non-inverted amplifier is required. In order to understand the operation of a non-inverted amplifier by an operational amplifier, some basic concepts are recalled below.
A scheme with the ideal operational amplifier is shown in the next figure.

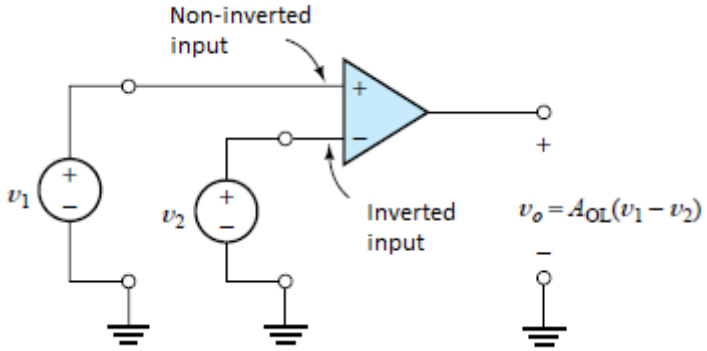


Figure 6.11. Scheme of an operational amplifier.

Basically an operational amplifier is a differential amplifier that presents a non-inverted and an inverted input. The input signals are denoted by $v_1(t)$ and $v_2(t)$ respectively. The subtraction between the input signals is called differential voltage and it is denoted by the following equation:

$$v_{id} = v_1 - v_2 \tag{5.3.1}$$

An ideal operational amplifier presents the following characteristics:

- Infinite input impedance.
- Infinite open loop gain, A_{OL}, for the differential signal.
- Zero output impedance.
- Infinite bandwidth.

The equivalent circuit of an ideal operational amplifier consists in a voltage generator dependent on the input signals. The open loop gain is very high, ideally infinite.

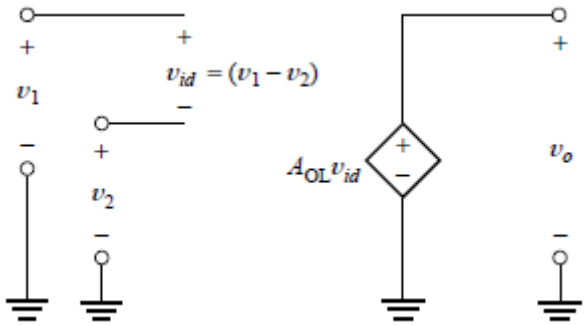


Figure 6.12. Equivalent circuit of the ideal operational amplifier.

Once the basic characteristics of an ideal operational amplifier have been defined, the next step is to know how to analyze this device in the configuration of a non-inverted amplifier. In general in the configuration of the operational amplifiers it is present negative feedback, in which part of the output signal comes back to the input in opposition to the input signal.
For an ideal operational amplifier the open loop gain is considered to be approximately infinite. For that reason a small differential voltage at the output results in a high output voltage. If there is negative feedback, part of the output signal comes back to the invert input terminal. This forces the input differential voltage to zero. If the open loop gain is considered to be infinite, then the input differential voltage is exactly zero. As the input differential voltage is zero, therefore the input current it is zero as well. The fact of forcing the input voltage and current to zero it is known as virtual short-circuit.

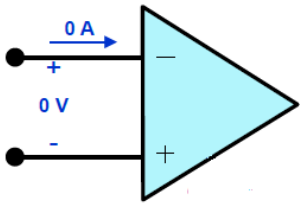


Figure 6.13. Virtual short-circuit condition.

Knowing the results of applying negative feedback, the analysis of ideal operational amplifiers is done following the next steps:

- Verify the presence of negative feedback, which is usually composed by several resistances connected to the output terminal and to the inverted input terminal.
- Consider that the input differential voltage and the input current at the input of the operational amplifier are zero (virtual short-circuit).

- Apply Kirchhoff laws and Ohm's law to obtain the voltage and current values of interest.

In the following figure it is shown the configuration of a non-inverted amplifier circuit.

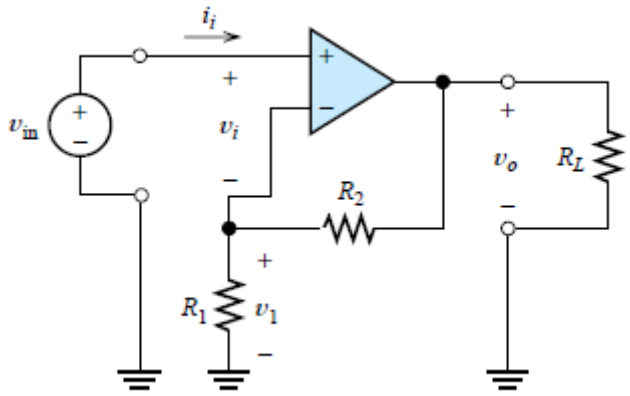


Figure 6.14. Schematic of a non-inverted amplifier.

In order to analyze the above circuit the above mentioned steps are followed. First of all it is considered that the operational amplifier is ideal. Then it is verified the presence of negative feedback. Assuming the voltage v_i is positive the output voltage is greater and positive as well. Part of the output tension goes to R_1because of the voltage divider formed by R_1 and R_2. The relation of the voltages at the input is expressed in the following equation:

$$v_i = v_{in} - v_1 \tag{5.3.2}$$

It can be noted that the voltage v_i decreases when v_o and v_1 increase. Therefore the network formed by both the amplifier and the feedback work to set the voltage v_i to zero ($v_i = 0$). The feedback is negative because of the feedback signal is opposed to the effect of the original input signal.
Once the presence of the negative feedback has been verified, the conditions for the virtual short-circuit can be applied to the circuit.

$$\begin{cases} v_i = 0 \\ i_i = 0 \end{cases} \tag{5.3.3}$$

Applying the virtual short-circuit condition to equation (5.3.2):

$$v_i = v_1 \tag{5.3.4}$$

As the input current i_i is equal to zero, then the voltage at R_1 is given by the expression of the voltage divider formed by R_1 and R_2.

$$v_1 = \frac{R_1}{R_1 + R_2} v_o \tag{5.3.5}$$

Substituting equation (5.3.5) into (5.3.4) yields:

$$v_i = \frac{R_1}{R_1 + R_2} v_o \tag{5.3.6}$$

The gain of the amplifier is defined as the ratio of the output voltage to the input voltage:

$$A_v = \frac{v_o}{v_i} \tag{5.3.7}$$

The resulting expression of the gain for the non-inverted amplifier using an ideal operational amplifier can be written as:

$$A_v = \frac{R_1 + R_2}{R_1} = 1 + \frac{R_2}{R_1} \tag{5.3.8}$$

It can be noted that the gain is positive, that is, there is no phase shift between the input and output signal. The gain depends on the values of the resistors present in the feedback network. The voltage gain is also independent on the value of the load impedance. Two more details to characterize the amplifier is that the input impedance is infinite because of the input current is zero. In addition the output impedance is zero. If we suppose an ideal operational amplifier then the circuit behaves as an ideal voltage amplifier.

Other aspect to consider is that

The ideal operational amplifier has been analyzed. But the real operational amplifier has to be analyzed as well. A real operational amplifier has to be supplied by one or two DC power supplies. The connections are showed in the following figure.

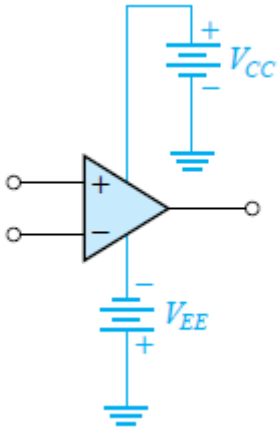


Figure 6.15. Operational amplifier with power supply connections.

It is important to mention that a real operational amplifier has two important limitations:

- Maximum output voltage: It is the maximum output voltage that the amplifier can provide to the output. It depends on the load impedance connected to the output. This allows to express the relationship between the input and output voltage. If the input signal is high enough to produce an output greater than the maximum output voltage, then the output experiences a cutting. This situation is showed in the following figure.

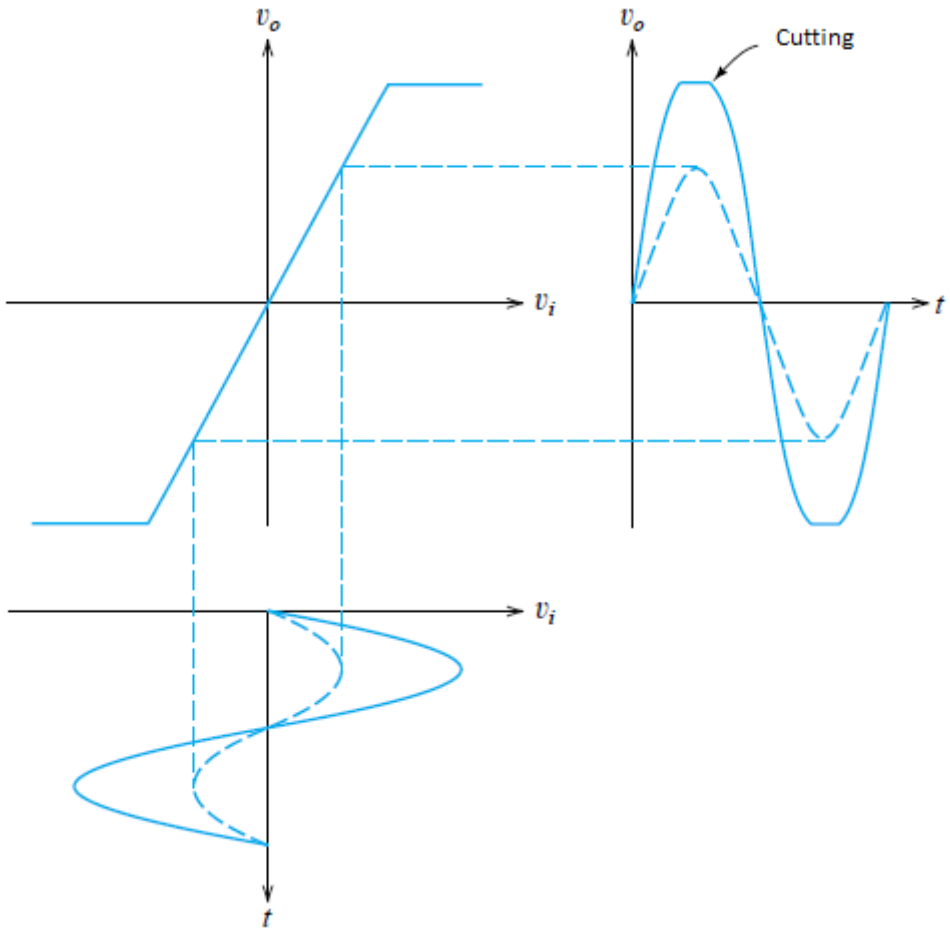


Figure 6.16. Limitations in a real operational amplifier.

- Maximum output current: Is the maximum output current that the operational amplifier can provide. If the load impedance need a current greater than the maximum value admitted the output signal experiences a cutting. The relationship between the currents is showed in the following figure.

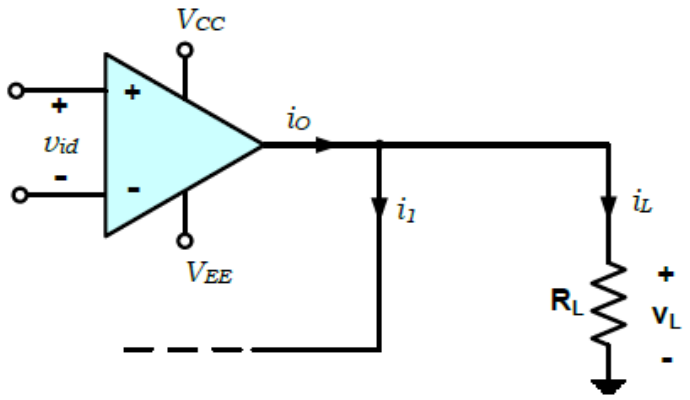


Figure 6.17. Current distribution at the output of the amplifier.

$$i_L = i_o - i_1 \tag{5.3.9}$$

Once it is known the behavior and limitations of a real operational amplifier, a non-inverted amplifier can be designed. The non-inverted amplifier is used to obtain voltages up to 10 V used to supply the phase shifters at the radar antenna. As the micro-controller Arduino Mega 2560 can generate voltages up to 5 V, it will be enough an amplifier with gain 2. In order to implement this non-inverted amplifier it will be used the operational amplifier μA741. This device works with a DC supply voltage of 15 V. The limitations of this device can be obtained from the datasheet.

- Maximum output voltage: $V_{oMAX} = \pm 14V$
- Maximum output current: $I_{oMAX} = \pm 25\ mA$

Then the next step is to choose the resistor values to obtain a gain of 2. The resistors used have a 5% of tolerance, therefore the gain will not be exactly 2 but it will be approximately 2. Using the expression of the closed loop gain of the non-inverted amplifier, in the equation (5.3.8) and setting it equal to 2, the relationship between the resistors is obtained:

$$1 + \frac{R_2}{R_1} = 2 \;\Rightarrow\; R_1 = R_2 \tag{5.3.10}$$

There are so many resistance values that could be used in this configuration to obtain the desired value of gain. However small values are not suitable, because the operational amplifier should provide at its output high values of current. In order to avoid this problems a particular case it is analyzed. If the value of resistors were 5 Ω, for an output voltage of 10V, then the current through the resistors would be 1 A.

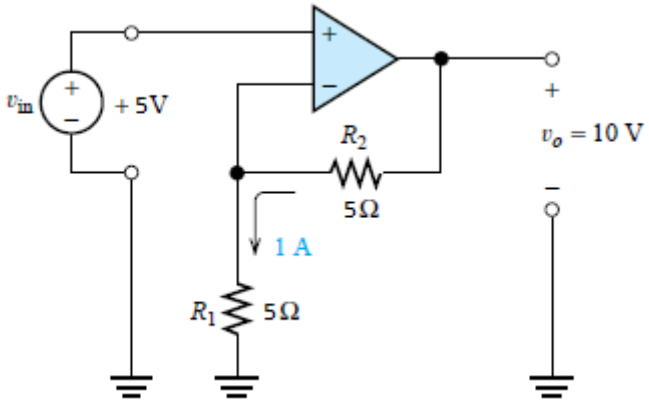


Figure 6.18. Non-inverted amplifier with small resistance values.

Generally most of the operational amplifiers cannot provide high values of current. In the case of using μA741 the maximum output current is $\pm 25\ mA$. If high values of resistors are used there, as example 5 MΩ is a non-desired effect called parasite effect. Circuits with high input impedance tend to take non-desired signals from other close circuits by parasite capacitive coupling.

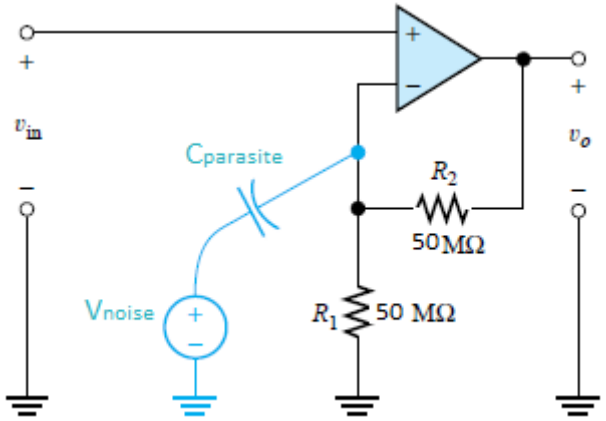


Figure 6.19. Non-inverted amplifier with high resistance values.

Therefore it is convenient to choose intermediate values of resistors to avoid these above mentioned effects. Finally the resistors chosen are:

$$R_1 = R_2 = 5.1k\Omega \tag{5.3.11}$$

With these resistor values, the current through the resistors is 1.37 mA, when the output voltage is the maximum. This value is smaller than the maximum output current that the operational amplifier can provide, therefore there is no limitation. Once the value of the resistances has been chosen, it is important to know if the limitation of the amplifier is

imposed by the maximum output current or the maximum output voltage. For this purpose the value of the equivalent DC impedance of the phase shifters used, has to be known. This can be obtained from the datasheet:

- Maximum Control Voltage Range: 13 V
- Maximum Control Current Range: 1 mA

Therefore by simply dividing these maximum values the DC impedance can be obtained:

$$R_L = 13\ k\Omega \tag{5.3.12}$$

Rewriting the equation (5.3.9), yields:

$$i_o = i_L + i_R \tag{5.3.13}$$

If the output voltage is the maximum:

$$i_L = \frac{V_{oMAX}}{R_L} = 1.077\ mA \tag{5.3.14}$$

$$i_R = \frac{V_{oMAX}}{R_1 + R_2} = 1.375\ mA \tag{5.3.15}$$

The maximum output current provided by the operational amplifier is:

$$i_{oMAX} = i_{LMAX} + i_R = 2.45\ mA < 25\ mA \tag{5.3.16}$$

Therefore there is no limitation due to the maximum output current. The limitation is imposed by the maximum output voltage. The maximum input will be $\pm 7V$, given by the equation (5.3.8).
The amplifiers designed work with DC signal. It is usually that real operational amplifiers present variations in their behavior working with DC signals. There are two types:

- Offset voltage
- Bias current

The offset voltage is the voltage, different from zero, present at the output of the operational amplifier, when the input is zero volts. This effect is due because of the current source formed by the transistors inside the operational amplifier have no perfect symmetry. This value also varies with temperature 10 μV/ºC.

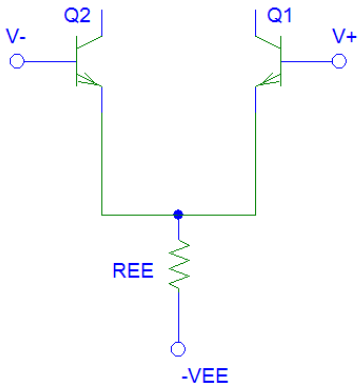


Figure 6.20. Current source within the real operational amplifier.

The offset voltage is not a fixed value. It is convenient to take some real measurements to determine the offset voltage. In this case when the input voltage is zero, a negative voltage at the output is obtained. Hence, the operational amplifier is considered ideal with the offset voltage modeled as a voltage source placed at any of the input, as depicted in the following figure.

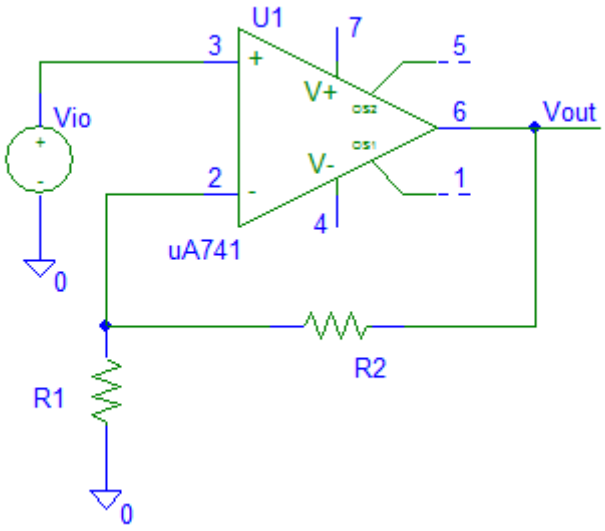


Figure 6.21. Scheme of the non-inverted amplifier with offset voltage.

The output voltage expression, when the input of the non-inverted amplifier is connected to ground, can be written as:

$$V_{in} = 0 \Rightarrow V_{out} = V_{io}\left(1 + \frac{R_2}{R_1}\right) \tag{5.3.17}$$

This value is different from zero, and could be significant if the gain is high. In this case the gain is not high, but a high accuracy is required to obtain the desired scan angle. Hence, the resources of the own device can be used in order to cancel the effects of the offset voltage. The operational amplifier used, µA741 DIP-8, has eight terminals as shown in the following figure.

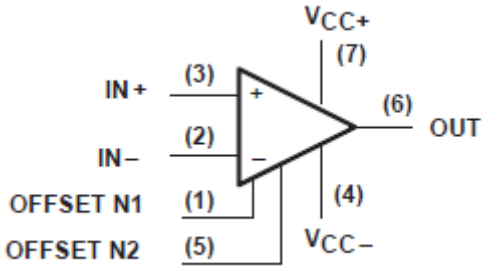


Figure 6.22. Scheme of the µA741 operational amplifier with all the terminals.

The terminals 1 and 5 are used to cancel the offset voltage. A variable resistor of 100kΩ is placed connecting both terminals. The value of the variable resistor is set to obtain the output voltage zero when the input voltage is also zero. This effect will not be considered in simulation results.

With relation to the offset bias current effect can be neglected, due to the fact that it produces variations of few mV.

Once the design it has finished, is important to simulate the amplifier in order to ensure that the circuit works as desired. There are several software programs to obtain the simulations. In this case Orcad Pspice 9.1 is used because of its simplicity. The schematic of the circuit is shown in the figure 6.19. As it can be noted the voltage generator used is sinusoidal. The reason is because a parametric simulation is required, where the input voltage takes values from 0 V to 5V. In order to see all the possible output values a sinusoidal generator is placed at the input with a frequency of 1000 Hz. Then after simulation a span of 0.5ms is set to show the results in the graph. The reason of choosing a span of 0.5ms is because for this purpose only positive values of the sinusoidal input voltage are displayed. Only positive values are showed because the DC voltages provided by analog ports of Arduino Mega 2560 will be positives as well. The power supplies chosen in order to supply the operational amplifier are symmetric:

$$V_{CC} = 15\ V \tag{5.3.18}$$

$$V_{EE} = -15\ V \tag{5.3.19}$$

Some capacitors are used in the design, to cancel any AC component or variation of the DC signal given by the power supplies.
The results of simulation are shown in the figure 6.20. Taking two different instants of time it can be noted that the gain of the amplifier is almost 2, therefore the circuit designed works as well as it is desired:

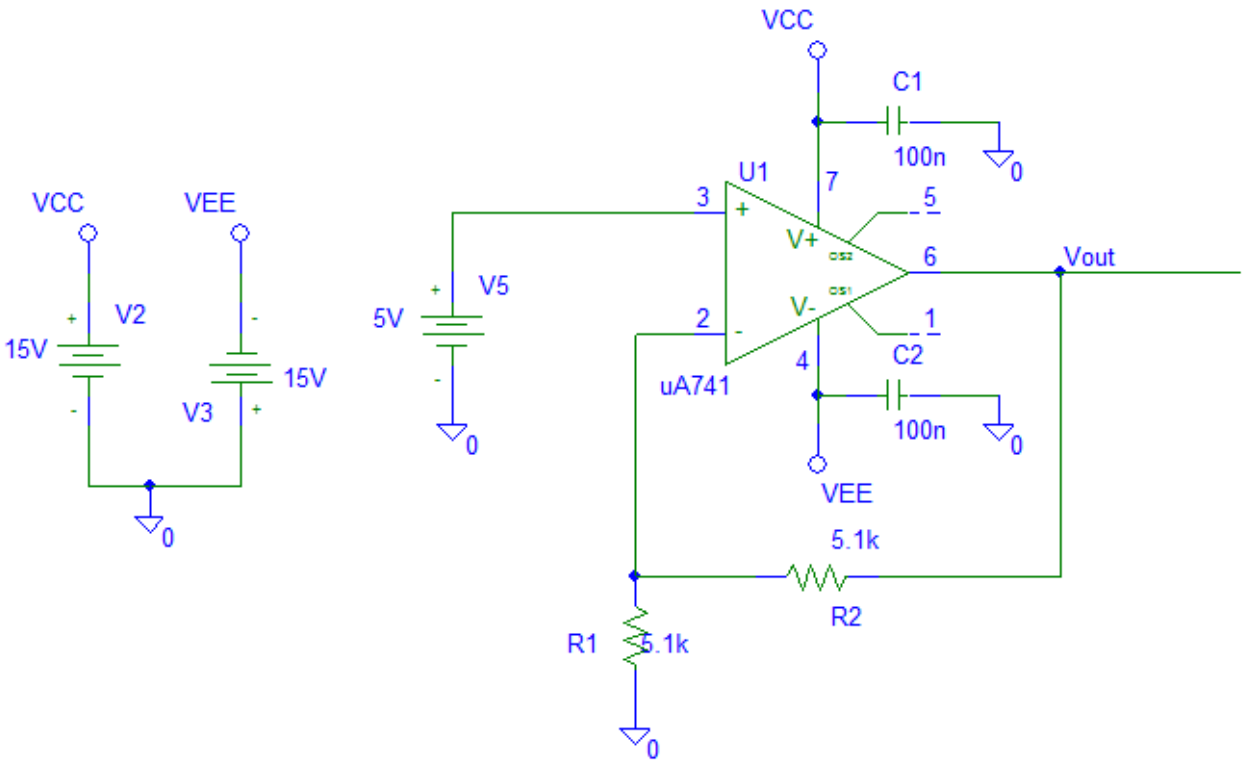


Figure 6.23. Schematic of the non-inverted amplifier.

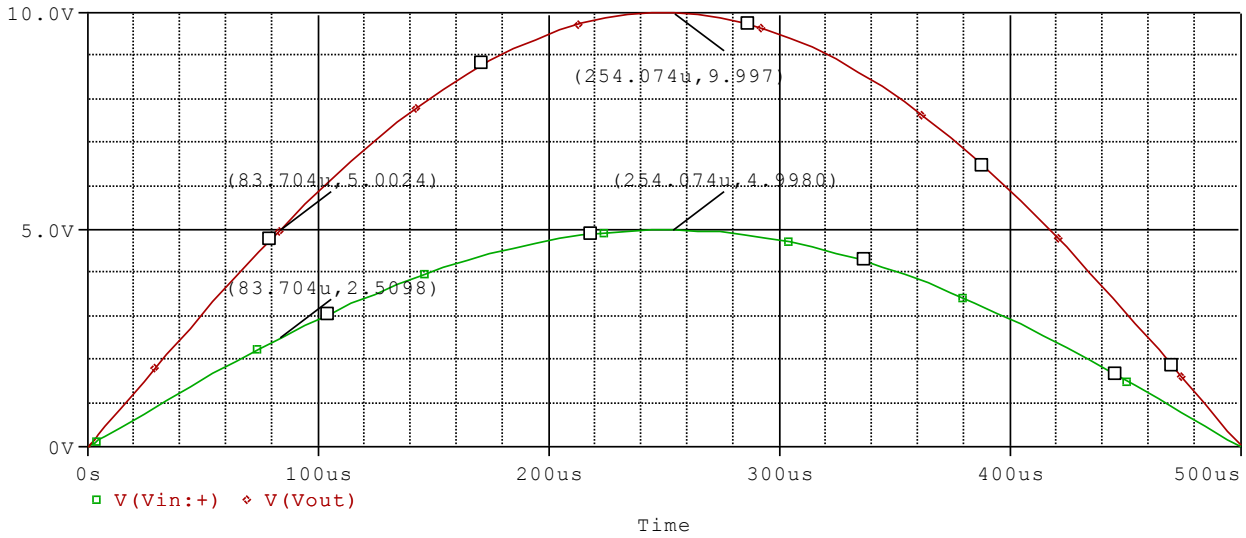


Figure 6.24. Output vs. input voltage of the non-inverted amplifier.

Once the amplifier stage has been designed and tested, the bias network design is finished. The software architecture used to implement the bias network of the phased array antenna is tested, with the scripts mentioned in section 6.3.1 and 6.3.2, in order to guarantee that the system works as desired. The software contains several functionalities which should be understandable for any user. There is no need for users to know MATLAB programming language to introduce the scan angle desired or test the phased array antenna. In order to make the software accessible is convenient to develop a graphical user interface.

6.5. Graphical User Interface

A graphical user interface (GUI), is a type of user interface that allows the users to interact with electronic devices or systems using images rather than text commands. It is important to develop an understandable interface, in order to allow the use of the system by any user. MATLAB software has a powerful tool called GUIDE, which is the acronym of Graphical User Interface Environment. GUIDE, at the same time, provides tools for designing user interfaces for custom applications. Using the GUIDE Layout Editor, a user interface can be designed. Then GUIDE automatically generates the MATLAB code for constructing the user interface. The graphical interface finality consists of obtaining the voltage values needed to change the directivity of the phased array and send it to the Arduino Mega 2560 micro-controller. This can be implemented by several options included in the interface:

- Calculate: The calculate option obtains the voltage values to be sent, from three input values: frequency, element spacing and scan angle desired. Although the system has been designed for a specific antenna at a frequency of 2.4 GHz, parameters such as frequency and element spacing can be set into different values. This allows using the interface for different phased arrays with the same number of radiating elements or less.

- Get Phases: This option only requires the phase shift as input parameter. Then it calculates the phases of each array element. In addition if the frequency and element spacing the interface shows the resulting scan angle.

- Get Voltages: This option calculate the corresponding voltages values to obtain in the phase shifters the phase shift desired.

- Send to Arduino: This option opens a serial communication between the host computer and the Arduino Board using the solution purposed with the MATLAB Support Package for Arduino. There is a small difference between the voltages set in the analog ports and the original voltages required to obtain the scan angle desired. This is due to the fact that the serial port sends bytes. One byte can represent integer values from 0 to 255. When the map function is used, it introduces a small error between the original voltage and the voltage set to the port. For this reason the interface prints out all the voltage values expected at the outputs of the amplifiers. In

order to obtain again voltages values from 0 to 10 volts, the function map.m must be used. As can be seen in the appendix B (B.6), the function map returns integer numbers. The interface must show the original voltages set with all decimal numbers. For this reason the function map is modified to map2.m , that makes the same that map.m but returns decimal numbers. The code of this new function is shown in the appendix B (B.7).

- Reset: This option is used to clean all the resulting boxes, except the input parameters. After executing this option, the analog ports of the micro-controller remain the same as before.

- Information: This remembers the users to connect the Arduino Board with the port 'COM4', before sending the voltages to the micro-controller.

The user interface is shown in the following figure.

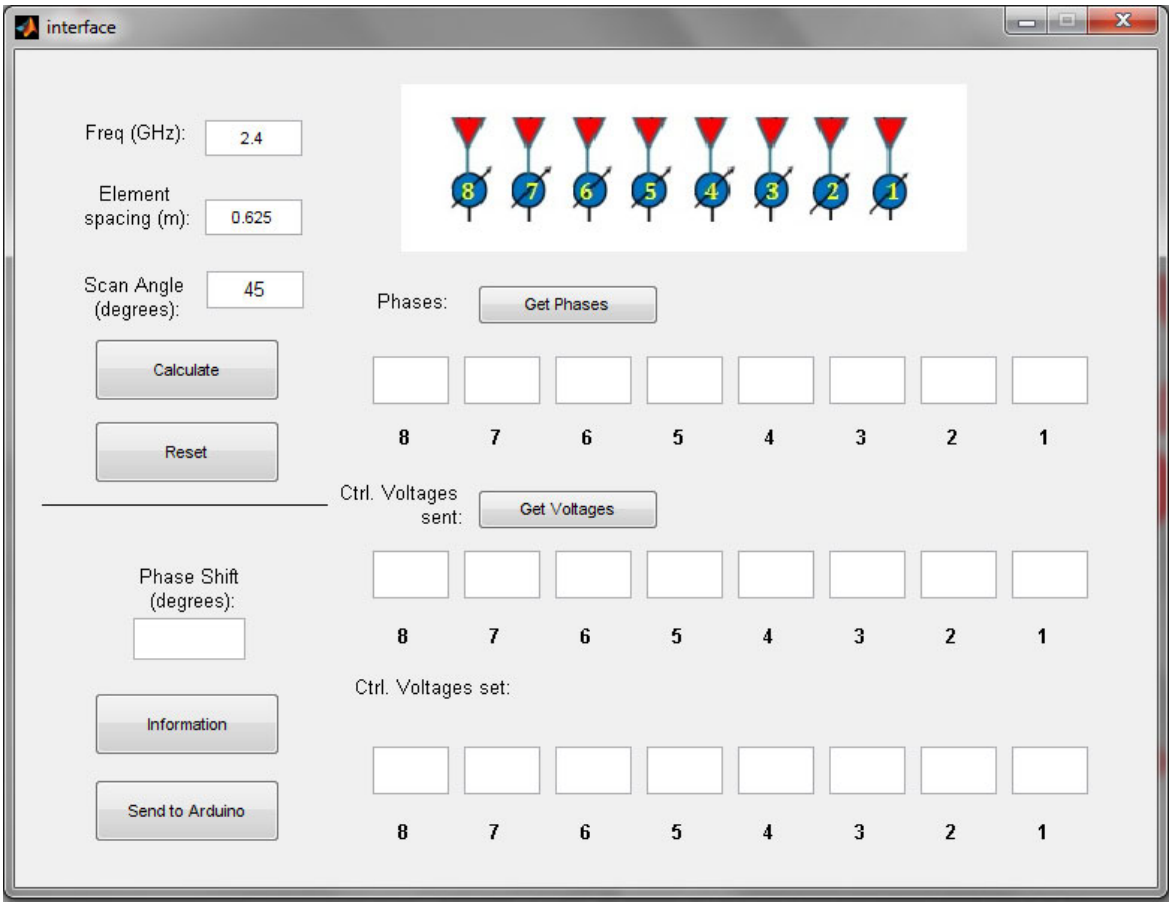


Figure 6.25. Graphical interface user of the bias network.

The user interface code is shown with detail in the appendix C (C.6).

7. CONCLUSION

In this study the feed and the bias network of a phased array antenna have been designed. It takes part of the development of a phased array radar. With reference to the feed network, the design is based on the simulations obtained by the software AWRDE 2011. The simulations are based on circuit analysis. This gives a first approximation of the final results. In order to implement the feed network an electromagnetic simulation is required. So the simulation provides results are close to the real measures.

Regarding the bias network design, the system is designed and implemented. The programs were implemented in several platforms. The software of the host computer was implemented in MATLAB, and the software of the Arduino was implemented using the Arduino environment. The software of the host computer consists of several functions working together in a script where a serial communication is created. Then the voltages are calculated from a scan angle and sent through the serial port to the Arduino micro-controller. In the study, two solutions are developed. The first one consists of a program in Arduino that receives the voltages values, represented by bytes, from the host computer and set the received values to the analog ports. The advantage using this solution is that the program on Arduino is very simple and the execution time is short. The antenna could change its directivity in a short period of time. This allows the radar making sweeps with different scan angles very fast and also allows the radar track targets in motion at high speed. On the other hand the second solution used the MATLAB Support Package for Arduino. Some of the advantages of this solution are that the program in Arduino has been developed previously and it catches all kind of errors sending data through the serial port. In addition it avoids some problems when using GUIs because of the synchronism that is present with the first solution. The execution time is longer than the first solution, but still allows the use of the radar in the applications commented before.

It is important to mention that there is a small error in the voltages at the output of the bias system. This is due to the fact that the micro-controller uses an integer number from 0 to 255 to set the voltages from 0 to 5 V at its ports. Therefore when the voltages values are converted from the interval [0,5] to [0,255], there is a small error introduced. This results in a small error in the obtained scan angle. There is one more small error introduced in the voltages controlling the phase shifters. The configuration of the amplifiers, designed to amplify the voltages provided by the micro-controller, uses operational amplifiers and the gain is function of the resistor values used in the design. The resistors used have a tolerance in its nominal value of the 5%. Therefore the resulting gain it is not exactly 2, but it should be approximately 2. This effect could introduce a small error in the scan angle of the phased array antenna.

Despite the small error introduced in the voltages, the solution works as desired, providing the scan angle from the host computer.

Appendix A : Fundamentals of electromagnetics

A.1 Notation for time-harmonic fields

Consider that the electric field of a time-harmonic plain wave travelling in the +z direction, only has x-component.

$$\vec{E}(\bar{r},t) = E_0 \cos(\omega t - kz)\,\hat{x} \tag{A.1.1}$$

The function cosine is an even function, therefore:

$$\cos(\theta) = \cos(-\theta) \tag{A.1.2}$$

Using this property the expression of the electric field can also be written as:

$$\vec{E}(\bar{r},t) = E_0 \cos(kz - \omega t)\,\hat{x} \tag{A.1.3}$$

Sometimes it is useful to obtain an equivalent expression of this field eliminating the time dependence. This is called phasor ($\vec{E}(\bar{r})$), and the relationship between the phasor and the time-harmonic expression is:

$$\vec{E}(\bar{r},t) = \mathbb{R}e\left\{\vec{E}(\bar{r})e^{-i\omega t}\right\} \tag{A.1.4}$$

$$\vec{E}(\bar{r},t) = \mathbb{R}e\left\{\vec{E}(\bar{r})e^{-i\omega t}\right\} = E_0 \cos(kz - \omega t)\,\hat{x} \tag{A.1.5}$$

$$\vec{E}(\bar{r},t) = \vec{E}(\bar{r},t) = \mathbb{R}e\left\{E_0 e^{ikz}\hat{x}e^{-i\omega t}\right\} \tag{A.1.6}$$

Then the phasor expression of a wave travelling in the positive z direction can be written as follows:

$$\vec{E}(\bar{r}) = E_0 e^{ikz}\hat{x} \tag{A.1.7}$$

Now we consider the electric field of a plane wave travelling in an arbitrary direction $\bar{r}$.

$$\vec{E}(\bar{r}) = E_0 e^{ik_x x} e^{ik_y y} e^{ik_z z} \hat{e} = E_0 e^{i(k_x x + k_y y + k_z z)} \hat{e} = E_0 e^{i\bar{k}\cdot\bar{r}} \hat{e} \tag{A.1.8}$$

$$\vec{E}(\bar{r}, t) = \mathbb{R}e\{\vec{E}(\bar{r}) e^{-i\omega t}\} = \mathbb{R}e\{E_0 e^{i\bar{k}\cdot\bar{r}} \hat{e} e^{-i\omega t}\} = E_0 \cos(\bar{k}\cdot\bar{r} - \omega t)\, \hat{e} \tag{A.1.9}$$

Where:

$$\vec{k} = k_x \hat{x} + k_y \hat{y} + k_z \hat{z} \text{ (Dispersion relation)} \tag{A.1.10}$$

$$\vec{r} = x\hat{x} + y\hat{y} + z\hat{z} \text{ (Position vector)} \tag{A.1.11}$$

From now on with the notation used:

- $\vec{E}(\bar{r}) = E_0 e^{i\bar{k}\cdot\bar{r}} \hat{e}$, is a wave travelling in the positive $\bar{r}$ direction.
- $\vec{E}(\bar{r}) = E_0 e^{-i\bar{k}\cdot\bar{r}} \hat{e}$, is a wave travelling in the negative $\bar{r}$ direction.

A.2 Maxwell's Equations

The Maxwell's equations are:

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \tag{A.2.1}$$

$$\nabla \times \vec{H} = \vec{J} + \frac{\partial \vec{D}}{\partial t} \tag{A.2.2}$$

$$\nabla \cdot \vec{D} = \rho \tag{A.2.3}$$

$$\nabla \cdot \vec{B} = 0 \tag{A.2.4}$$

In order to express Maxwell's equations using phasor notation, it has to be noted that:

$$\frac{\partial}{\partial t}\vec{E}(\bar{r}, t) = \frac{\partial}{\partial t}\mathbb{R}e\{\vec{E}(\bar{r}) e^{-i\omega t}\} = \mathbb{R}e\left\{\vec{E}(\bar{r})\frac{\partial}{\partial t} e^{-i\omega t}\right\} \tag{A.2.5}$$

$$\frac{\partial}{\partial t} e^{-i\omega t} = -i\omega e^{-i\omega t} \tag{A.2.6}$$

$$\frac{\partial}{\partial t}\vec{E}(\bar{r},t) = \mathbb{R}e\{-i\omega\vec{E}(\bar{r})e^{-i\omega t}\} = \mathbb{R}e\left\{\frac{\partial}{\partial t}\vec{E}(\bar{r})e^{-i\omega t}\right\} \tag{A.2.7}$$

The derivative of the phasor of a time-harmonic field can be written as:

$$\frac{\partial}{\partial t}\vec{E}(\bar{r}) = -i\omega\vec{E}(\bar{r}) \tag{A.2.8}$$

Finally Maxwell's equations become:

$$\nabla \times \vec{E} = i\omega\vec{B} \tag{A.2.9}$$

$$\nabla \times \vec{H} = \vec{J} - i\omega\vec{D} \tag{A.2.10}$$

$$\nabla \cdot \vec{D} = \rho \tag{A.2.11}$$

$$\nabla \cdot \vec{B} = 0 \tag{A.2.12}$$

A.3 Solution equations to radiated time-harmonic fields

Using the Maxwell equation (A.2.12) in time-harmonic form we note that $\vec{B}$ is divergence-less field and by Hemholtz's theorem it can be expressed by the curl of a magnetic potential vector:

$$\nabla \times \vec{A} = \vec{B} \tag{A.3.1}$$

Considering a simple (linear, isotropic, homogeneous, and non-dispersive) medium, the constitutive relations are:

$$\vec{D} = \epsilon\vec{E} \tag{A.3.2}$$

$$\vec{B} = \mu\vec{H} \tag{A.3.3}$$

Applying equation (A.3.1) to Maxwell-Faraday equation (A.2.2) the following expression is obtained:

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} = -\frac{\partial}{\partial t}\nabla \times \vec{A} = -\nabla \times \frac{\partial}{\partial t}\vec{A} \tag{A.3.4}$$

Therefore moving the right hand side of the equation to the left, we obtain a free-curl vector:

$$\nabla \times \left(\vec{E} + \frac{\partial}{\partial t}\vec{A}\right) = 0 \tag{A.3.5}$$

Again by Hemholtz's theorem this vector can be expressed as the gradient of a scalar potential. Finally the expression of the electric field is obtained:

$$\vec{E} + \frac{\partial}{\partial t}\vec{A} = -\nabla V \;\Rightarrow\; \vec{E} = -\nabla V - \frac{\partial}{\partial t}\vec{A} \tag{A.3.6}$$

Now to obtain the wave equation radiated, we take the curl of the Maxwell-Ampere's equation:

$$\nabla \times \nabla \times \vec{A} = \mu\vec{J} + \mu\epsilon\frac{\partial}{\partial t}\vec{E} = \nabla\left(\nabla \cdot \vec{A}\right) - \nabla^2\vec{A} \tag{A.3.7}$$

Substituting the electric field into the last equation:

$$\nabla\left(\nabla \cdot \vec{A}\right) - \nabla^2\vec{A} = \mu\vec{J} - \mu\epsilon\frac{\partial}{\partial t}(\nabla V + \frac{\partial}{\partial t}\vec{A})$$

$$\nabla^2\vec{A} - \mu\epsilon\frac{\partial^2}{\partial t^2}\vec{A} = -\mu\vec{J} + \nabla(\nabla \cdot \vec{A} + \mu\epsilon\frac{\partial}{\partial t}V) \tag{A.3.8}$$

Applying Lorentz's condition:

$$\nabla \cdot \vec{A} + \mu\epsilon\frac{\partial}{\partial t}V = 0 \tag{A.3.9}$$

$$\nabla \cdot \vec{A} - i\omega\mu\epsilon V = 0 \tag{A.3.10}$$

Wave equation becomes:

$$\nabla^2\vec{A} - \mu\epsilon\frac{\partial^2}{\partial t^2}\vec{A} = -\mu\vec{J} \tag{A.3.11}$$

It can also be written using phasor notation:

$$\nabla^2 \vec{A} + k^2 \vec{A} = -\mu \vec{J} \tag{A.3.12}$$

Where k is the dispersion relation:

$$k = \omega\sqrt{\mu\epsilon} \tag{A.3.13}$$

The solution to the wave equation is:

$$\vec{A}(\bar{r}) = \frac{\mu}{4\pi} \iiint \frac{\vec{J}(\bar{r}')}{|\bar{r} - \bar{r}'|} e^{ik|\bar{r}-\bar{r}'|}\, dV' \tag{A.3.14}$$

Once obtained the expression of the magnetic vector potential the expressions of fields can be obtained by:

$$\vec{H} = \frac{1}{\mu} \nabla \times \vec{A} \tag{A.3.15}$$

$$\vec{E} = -\nabla V + i\omega \vec{A} \tag{A.3.16}$$

Substituting the Lorentz's condition into the second equation:

$$\vec{E} = i\omega \vec{A} - \nabla \frac{\nabla \cdot \vec{A}}{i\omega\mu\epsilon} \tag{A.3.17}$$

In the far-field the effect of the second term of the equation is neglected:

$$\vec{E} = i\omega \vec{A} \tag{A.3.18}$$

Then to calculate the far-field expressions first of all we obtain the expression of the magnetic vector potential and to finish we derive the expressions of fields using:

$$\vec{H} = \frac{1}{\mu} \nabla \times \vec{A} \tag{A.3.19}$$

Another way to obtain the electric field is using Maxwell-Ampere's law with phasors:

$$\nabla \times \vec{H} = \vec{J} - i\omega\epsilon\vec{E} \tag{A.3.20}$$

In the far field, there are no current sources:

$$\vec{J} = 0 \;\Rightarrow\; \nabla \times \vec{H} = -i\omega\epsilon\vec{E} \tag{A.3.21}$$

$$\vec{E} = -\frac{\nabla \times \vec{H}}{i\omega\epsilon} \tag{A.3.22}$$

A.4 Vector derivatives applied to radiation

The magnetic vector potential is obtained in spherical coordinates, so to obtain the electric and magnetic field the curl operator has to be calculated. The expression of the curl of a vector field $\vec{A}$ in spherical coordinates is:

$$\nabla \times \vec{A} = \frac{1}{r\sin\theta}\left[\frac{\partial}{\partial\theta}\left(A_\phi \sin\theta\right) - \frac{\partial A_\theta}{\partial\phi}\right]\hat{r} + \frac{1}{r}\left[\frac{1}{\sin\theta}\frac{\partial A_r}{\partial\phi} - \frac{\partial}{\partial r}\left(rA_\phi\right)\right]\hat{\theta} + \frac{1}{r}\left[\frac{\partial}{\partial r}\left(rA_\theta\right) - \frac{\partial A_r}{\partial\theta}\right]\hat{\phi} \tag{A.3.23}$$

Appendix B : MATLAB functions

In this appendix is shown the code of the functions used to calculate the control voltages to be sent, corresponding to a scan angle generated.

B.1. Function generate_scan

```
% This function calculates a random beam steering or scan angle with
% uniform distribution function from -90 to 90 degrees.
%
% OUTPUT PARAMETERS:
% scan_ang: Scan  angle of the antenna.
function [scan_ang]=generate_scan()

scan_ang=unifrnd(-90,90,1,1);

end
```

B.2. Function calculate_phaseShift

```
% This function calculates the phase shift beetween two array elements
% to obtain the beam steering introduced as input argument.
%
% INPUT PARAMETERS:
% freq: Frequency of signal.
% d: Distance between two array elements.
% scan_ang: Scan angle.
%
% OUTPUT PARAMETERS:
% phase_sf: Phase shift between two array elements.
%
function [phase_shift]=calculate_phaseShift(freq,d,scan_ang)

c=3*10^8;   %Speed of light
lambda=c/freq;    %Wave length
x=d*sind(abs(scan_ang));
phase_shift=(360*x)/lambda; %Phase shift between array elements

end
```

B.3. Function calculate_phases

```
% This function calculates a vector with all the current phases needed
% to set in each array element to get the beam steering desired.
%
% INPUT PARAMETERS:
% N: Number of array elements.
% phase_shift: Phase shift between two consecutive array elements to obtain
% the corresponding scan angle.
%
% OUTPUT PARAMETERS:
```

```
% phases: Vector with the phases of each array element.
%
function [phases]=calculate_phases(N,phase_shift)

for i=0:N-1
    ph=phase_shift*i;
    if ph>=360
        n=floor(ph/360);
        ph=ph-n*360;  % The phase is expressed with values from 0 to 360 degrees
    end
    phases(1,i+1)=ph;
end
```

B.4. Function toggle

```
% This function inverts the order of the elements of a vector.
%
% INPUT PARAMETERS:
% vector1: Vector with the value of phases.
%
% OUTPUT PARAMETERS:
% vector2: Vector with order of elements inverted.
%
function [vector2]=toggle(vector1)

j=length(vector1);
for i=1:length(vector1)
    vector2(i)=vector1(j);
    j=j-1;
end
```

B.5. Function get_voltages

```
% This function calculates the voltages that should be applied to the phase
% shifters in order to obtain the phases, introduced as input parameter,
% needed to generate a specific scan angle.
%
% INPUT PARAMETERS:
% phases: Matrix with the phases should be applied to each array element.
%
% OUTPUT PARAMETERS:
% voltages: Matrix with the voltages should be applied to each phase
% shifter to obtain the corresponding phases.
%
function [voltages]=get_voltages(phases)

% X coordinate vector containing the points obtained from the control voltage
versus phase shift graph by the function graph_picker.
X=[0.000000e+000 7.449857e-002 1.489971e-001 2.234957e-001 3.166189e-001
4.097421e-001 5.214900e-001 6.332378e-001 7.263610e-001 8.194842e-001 9.498567e-
001 1.061605e+000 1.191977e+000 1.303725e+000 1.434097e+000 1.564470e+000
1.694842e+000 1.825215e+000 1.955587e+000 2.085960e+000 2.197708e+000
2.346705e+000 2.477077e+000 2.626074e+000 2.775072e+000 2.924069e+000
3.073066e+000 3.203438e+000 3.333811e+000 3.501433e+000 3.631805e+000
3.780802e+000 3.911175e+000 4.041547e+000 4.190544e+000 4.339542e+000
4.507163e+000 4.656160e+000 4.805158e+000 4.935530e+000 5.065903e+000
5.214900e+000 5.345272e+000 5.494269e+000 5.624642e+000 5.755014e+000
```

```
5.885387e+000 6.053009e+000 6.183381e+000 6.332378e+000 6.481375e+000
6.611748e+000 6.760745e+000 6.928367e+000 7.095989e+000 7.263610e+000
7.412607e+000 7.561605e+000 7.729226e+000 7.878223e+000 8.027221e+000
8.176218e+000 8.325215e+000 8.492837e+000 8.641834e+000 8.828080e+000
8.995702e+000 9.126074e+000 9.275072e+000 9.424069e+000 9.554441e+000
9.703438e+000 9.871060e+000 1.002006e+001 1.018768e+001 1.031805e+001
1.048567e+001 1.061605e+001 1.076504e+001 1.089542e+001 1.104441e+001
1.117479e+001 1.130516e+001 1.141691e+001 1.154728e+001 1.167765e+001
1.180802e+001 1.193840e+001 1.203152e+001 1.216189e+001 1.232951e+001
1.245989e+001 1.259026e+001 1.272063e+001 1.285100e+001 1.296275e+001 ];

% Y coordinate vector containing the points obtained from the control voltage
% versus phase shift graph by the function graph_picker.
Y=[0.000000e+000 5.586592e+000 1.256983e+001 1.815642e+001 2.653631e+001
3.491620e+001 4.189944e+001 5.027933e+001 5.726257e+001 6.424581e+001
7.122905e+001 7.960894e+001 8.659218e+001 9.357542e+001 1.019553e+002
1.075419e+002 1.145251e+002 1.229050e+002 1.284916e+002 1.340782e+002
1.396648e+002 1.466480e+002 1.536313e+002 1.592179e+002 1.648045e+002
1.703911e+002 1.759777e+002 1.815642e+002 1.871508e+002 1.927374e+002
1.997207e+002 2.025140e+002 2.081006e+002 2.136872e+002 2.178771e+002
2.234637e+002 2.276536e+002 2.332402e+002 2.374302e+002 2.430168e+002
2.472067e+002 2.513966e+002 2.555866e+002 2.611732e+002 2.653631e+002
2.695531e+002 2.751397e+002 2.793296e+002 2.863128e+002 2.905028e+002
2.946927e+002 2.988827e+002 3.044693e+002 3.100559e+002 3.156425e+002
3.212291e+002 3.268156e+002 3.324022e+002 3.365922e+002 3.421788e+002
3.477654e+002 3.505587e+002 3.561453e+002 3.603352e+002 3.659218e+002
3.701117e+002 3.756983e+002 3.798883e+002 3.840782e+002 3.882682e+002
3.938547e+002 3.994413e+002 4.022346e+002 4.078212e+002 4.120112e+002
4.162011e+002 4.203911e+002 4.245810e+002 4.287709e+002 4.329609e+002
4.371508e+002 4.413408e+002 4.441341e+002 4.469274e+002 4.511173e+002
4.553073e+002 4.581006e+002 4.608939e+002 4.636872e+002 4.664804e+002
4.706704e+002 4.734637e+002 4.762570e+002 4.790503e+002 4.818436e+002
4.832402e+002 ];

for i=1:length(phases)
    voltages(i)=INTERP1(Y,X,phases(i),'spline');
end
```

B.6. Function map

```
% This function converts one value within the range [fromLow,fromHigh]
% to the corresponding value within the range [toLow,toHigh].
%
% INPUT PARAMETERS:
% val: Value which it is going to be converted.
% fromLow: Minimum value of the origin range.
% fromHigh: Maximum value of the origin range.
% toLow: Minimum value of the destination range.
% toHigh: Maximum value of the destination range.
%
% OUTPUT PARAMETERS:
% num: Converted integer value of val within the range [toLow,toHigh]
function [num]=map(val,fromLow,fromHigh,toLow,toHigh)

aux1=val/(fromHigh-fromLow);
aux2=aux1*(toHigh-toLow);
num=floor(aux2);

end
```

B.7. Function map2

```
% This function converts one value within the range [fromLow,fromHigh]
% to the corresponding value within the range [toLow,toHigh].
%
% INPUT PARAMETERS:
% val: Value which it is going to be converted.
% fromLow: Minimum value of the origin range.
% fromHigh: Maximum value of the origin range.
% toLow: Minimum value of the destination range.
% toHigh: Maximum value of the destination range.
%
% OUTPUT PARAMETERS:
% num: Converted decimal value of val within the range [toLow,toHigh]
function [num]=map2(val,fromLow,fromHigh,toLow,toHigh)

aux1=val/(fromHigh-fromLow);
aux2=aux1*(toHigh-toLow);
num= aux2;

end
```

Appendix C : Arduino software, test scripts and GUI

C.1. Arduino software set_voltages.pde

In this appendix section it is shown the program developed to receive the bytes representing the voltage values to be set at the analog output ports of the Arduino Mega 2560.

```
int incomingByte = 0;   // byte read
int portNum;

void setup() {
        Serial.begin(9600);   // open serial port at 9600 bps
        // the ports used are from 2 to 9
        pinMode(2,OUTPUT);
        pinMode(3,OUTPUT);
        pinMode(4,OUTPUT);
        pinMode(5,OUTPUT);
        pinMode(6,OUTPUT);
        pinMode(7,OUTPUT);
        pinMode(8,OUTPUT);
        pinMode(9,OUTPUT);

        portNum=2;
        for(int i=2;i<10;i++){ //initialize all the analog ports used to 0 volts
          analogWrite(i,0);
        }
}

void loop() {

      // only if there is incoming data
      if (Serial.available() > 0) {
                if(portNum>9){
                  portNum=2;
                }
                // read the incoming byte
            incomingByte = Serial.read();
                // assign the byte read with the corresponding port
                analogWrite(portNum,incomingByte);
                portNum++;
                delay(100);
      }
}
```

C.2. Script software1

This Matlab script calculates and sends the voltage values, codified in bytes, through the serial port COM4 to the micro-controller Arduino Mega 2560. The voltages are calculated from the scan angle desired, introduced by the keyboard. The code of this script is as follows:

```
clear all
clc
%Create a serial connection
serialPort=serial('COM4','BaudRate',9600);
%Inform about errors
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
%Open the serial port
fopen(serialPort);

%Frequency of the signal
f=2.4*10^9;
%Speed of light
c=3*10^8;
%Wavelength
lambda=c/f;
%Distance between two consecutive elements
d=lambda/2;
%Number of array elements
N=8;

option=input('Introduce scan angle or esc to exit:','s');
while strcmp(option,'esc')==0
    scan_ang=str2double(option);
    while (scan_ang<-90 || scan_ang>90) && strcmp(option,'esc')==0
        disp('Value outside the range. Please introduce a value within the range
[-90,90]');
        disp('Press any key to continue.');
        pause
        clc
        option=input('Introduce scan angle or esc to exit:','s');
        scan_ang=str2double(option);
    end
    if strcmp(option,'esc')==0
        %Calculate the phase shift needed to obtain the scan angle
        phase_shift=calculate_phaseShift(lambda,d,scan_ang);
        %Print the phase shift
        disp(['phase shift = ' num2str(phase_shift)]);
        %Calculate the phase of each array element
        phases=calculate_phases(N,phase_shift);
        %If the scan angle is negative change the order of phases
        if(scan_ang<0)
            phases=toggle(phases);
        end
        %Print the phases of each array element
        disp(['phases = ' num2str(phases)]);
        %Calculate the corresponding control voltages to each phase
        for i=1:length(phases)
            voltages(i)=get_voltage(phases(i));
        end
        %Print all the voltage values should be applied to each phase shifter
        disp(['voltages = ' num2str(voltages)]);
        %Voltages are mapped from 0-10 to 0-255
```

```
        for i=1:length(voltages)
            voltages_mapped(i)=map(voltages(i),0,10,0,255);
        end
        %Voltage values are sent via serial port.
        for i=1:length(voltages_mapped)
            fwrite(serialPort,voltages_mapped(i),'uint8');
        end
        disp('Press any key to continue.');
        pause
        clc
        option=input('Introduce scan angle or esc to exit:','s');
    end
end
%Close the serial port
fclose(serialPort);
clear all
clc
```

C.3. Script software2

This Matlab script calculates and sends to Arduino the voltage values needed to obtain a scan angle generated by the function generate_scan defined in the section 6.2 of chapter 6. The code of this script can be shown below.

```
clear all
clc
%Create a serial connection
serialPort=serial('COM4','BaudRate',9600);
%Inform about errors
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
%Open the serial port
fopen(serialPort);

%Frequency of the signal
f=2.4*10^9;
%Speed of light
c=3*10^8;
%Wavelength
lambda=c/f;
%Distance between two consecutive elements
d=lambda/2;
%Number of array elements
N=8;

option='Y';
while strcmp(option,'Y')==1
    clc
    %Generate the a scan angle from -90 to 90 degrees
    scan_ang=generate_scan();
    %Print the scan angle generated
    disp(['scan angle = ' num2str(scan_ang)]);
    %Calculate the phase shift needed to obtain the scan angle
    phase_shift=calculate_phaseShift(lambda,d,scan_ang);
    %Print the phase shift
    disp(['phase shift = ' num2str(phase_shift)]);
    %Calculate the phase of each array element
    phases=calculate_phases(N,phase_shift);
```

```
    %If the scan angle is negative change the order of phases
    if(scan_ang<0)
        phases=toggle(phases);
    end
    %Print the phases of each array element
    disp(['phases = ' num2str(phases)]);
    %Calculate the corresponding control voltages to each phase
    for i=1:length(phases)
        voltages(i)=get_voltage(phases(i));
    end
    %Print all the voltage values should be applied to each phase shifter
    disp(['voltages = ' num2str(voltages)]);
    %Voltages are mapped from 0-10 to 0-255
    for i=1:length(voltages)
        voltages_mapped(i)=map2(voltages(i),0,10,0,255);
    end
    %Voltage values are sent via serial port.
    for i=1:length(voltages_mapped)
        fwrite(serialPort,voltages_mapped(i),'uint8');
    end
    %The program asks for continue or stop the running
    option=input('Do you want to continue? Y/N [Y]: ','s');
    if isempty(option)
        option='Y';
    end
end
%Close the serial port
fclose(serialPort);
clear all
clc
```

.

C.4. Script software3

This MATLAB script has the same functionality as the script software1 but uses the MATLAB Support Package for Arduino.

```
clear all
clc
%Create a serial connection
serialPort=arduino('COM4');
%Initializing the control voltages to zero
for portNum=6:13
    serialPort.analogWrite(portNum,0);
end
%Frequency of the signal
f=2.4*10^9;
%Speed of light
c=3*10^8;
%Wavelength
lambda=c/f;
%Distance between two consecutive elements
d=lambda/2;
%Number of array elements
N=8;

option=input('Introduce scan angle or esc to exit:','s');
while strcmp(option,'esc')==0
    scan_ang=str2double(option);
```

```
    while (scan_ang<-90 || scan_ang>90) && strcmp(option,'esc')==0
        disp('Value outside the range. Please introduce a value within the range
[-90,90]');
        disp('Press any key to continue.');
        pause
        clc
        option=input('Introduce scan angle or esc to exit:','s');
        scan_ang=str2double(option);
    end
    if strcmp(option,'esc')==0
        %Calculate the phase shift needed to obtain the scan angle
        phase_shift=calculate_phaseShift(lambda,d,scan_ang);
        %Print the phase shift
        disp(['phase shift = ' num2str(phase_shift)]);
        %Calculate the phase of each array element
        phases=calculate_phases(N,phase_shift);
        %If the scan angle is negative change the order of phases
        if(scan_ang<0)
            phases=toggle(phases);
        end
        %Print the phases of each array element
        disp(['phases = ' num2str(phases)]);
        %Calculate the corresponding control voltages to each phase
        warning('off','MATLAB:dispatcher:InexactCaseMatch');
        for i=1:length(phases)
            voltages(i)=get_voltage(phases(i));
        end
        %Print all the voltage values should be applied to each phase shifter
        disp(['voltages = ' num2str(voltages)]);
        %Voltages are mapped from 0-10 to 0-255
        for i=1:length(voltages)
            voltages_mapped(i)=map(voltages(i),0,10,0,255);
        end
        %Voltage values are sent via serial port.
        portNum=2;
        for i=1:length(voltages_mapped)
            serialPort.analogWrite(portNum,voltages_mapped(i));
            portNum=portNum+1;
        end
        disp('Press any key to continue');
        pause
        clc
        option=input('Introduce scan angle or esc to exit:','s');
    end
end
%Close the serial port
delete(serialPort);
clear all
clc
```

C.5. Script software4

This MATLAB script has the same functionality as the script software2 but uses the MATLAB Support Package for Arduino.

```
clear all
clc
%Creates a serial connection
serialPort=arduino('COM4');
%Frequency of the signal
f=2.4*10^9;
%Speed of light
c=3*10^8;
%Wavelength
lambda=c/f;
%Distance between two consecutive elements
d=lambda/2;
%Number of array elements
N=8;

option='Y';
while strcmp(option,'Y')==1
    clc
    scan_ang=generate_scan();
    disp(['scan angle = ' num2str(scan_ang)]);
    phase_shift=calculate_phaseShift(lambda,d,scan_ang);
    disp(['phase shift = ' num2str(phase_shift)]);
    phases=calculate_phases(N,phase_shift);
    if(scan_ang<0)
        phases=toggle(phases);
    end

    disp(['phases = ' num2str(phases)]);
    for i=1:length(phases)
        voltages(i)=get_voltage(phases(i));
    end
    disp(['voltages = ' num2str(voltages)]);
    for i=1:length(voltages)
        voltages_mapped(i)=map2(voltages(i),0,10,0,255);
    end

    portNum=2;
    for i=1:length(voltages_mapped)
        serialPort.analogWrite(portNum,voltages_mapped(i));
        portNum=portNum+1;
    end

    option=input('Do you want to continue? Y/N [Y]: ','s');
    if isempty(option)
        option='Y';
    end
end

delete(serialPort);
clear all
clc
```

C.6. Bias Network GUI

Here is the code of the graphical user interface of the software controlling the bias network design. The code has to be copied and saved in an .m file in order to execute the software in any host computer. As this GUI uses the functions defined in the section 6.2, the interface file has to be in a folder with the files corresponding with all the functions used. In addition the GUI uses the MATLAB Support Package for Arduino, for that reason the folder have to contain all the files of this package as well. The code of the GUI is as follows:

```
function varargout = phasedArray(varargin)
% PHASEDARRAY M-file for phasedArray.fig
%      PHASEDARRAY, by itself, creates a new PHASEDARRAY or raises the existing
%      singleton*.
%
%      H = PHASEDARRAY returns the handle to a new PHASEDARRAY or the handle to
%      the existing singleton*.
%
%      PHASEDARRAY('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PHASEDARRAY.M with the given input arguments.
%
%      PHASEDARRAY('Property','Value',...) creates a new PHASEDARRAY or raises
the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before phasedArray_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to phasedArray_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help phasedArray

% Last Modified by GUIDE v2.5 24-Jun-2013 11:27:19

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @phasedArray_OpeningFcn, ...
                   'gui_OutputFcn',  @phasedArray_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before phasedArray is made visible.
function phasedArray_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to phasedArray (see VARARGIN)

% Choose default command line output for phasedArray
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes phasedArray wait for user response (see UIRESUME)
% uiwait(handles.figure1);
clc
clear serialPort;
global serialPort;
serialPort=arduino('COM4');

% Initialize all the analog ports of the micro-controller with zero volts
for portNum=6:13
    serialPort.analogWrite(portNum,0);
end

% --- Outputs from this function are returned to the command line.
function varargout = phasedArray_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes during object creation, after setting all properties.
function figure_CreateFcn(hObject, eventdata, handles)
% hObject    handle to figure (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
image=imread(strcat('C:\Users\Gabri\Desktop\arduino\','figure.jpg'));
imshow(image);
guidata(hObject,handles)
% Hint: place code in OpeningFcn to populate figure


function freq_Callback(hObject, eventdata, handles)
% hObject    handle to freq (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of freq as text
%        str2double(get(hObject,'String')) returns contents of freq as a double


% --- Executes during object creation, after setting all properties.
function freq_CreateFcn(hObject, eventdata, handles)
% hObject    handle to freq (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function elemSpacing_Callback(hObject, eventdata, handles)
% hObject    handle to elemSpacing (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of elemSpacing as text
%        str2double(get(hObject,'String')) returns contents of elemSpacing as a
double


% --- Executes during object creation, after setting all properties.
function elemSpacing_CreateFcn(hObject, eventdata, handles)
% hObject    handle to elemSpacing (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function scanAngle_Callback(hObject, eventdata, handles)
% hObject    handle to scanAngle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of scanAngle as text
%        str2double(get(hObject,'String')) returns contents of scanAngle as a
double

% --- Executes during object creation, after setting all properties.
function scanAngle_CreateFcn(hObject, eventdata, handles)
% hObject    handle to scanAngle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function phaseShift_Callback(hObject, eventdata, handles)
% hObject    handle to phaseShift (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of phaseShift as text
%        str2double(get(hObject,'String')) returns contents of phaseShift as a
double


% --- Executes during object creation, after setting all properties.
function phaseShift_CreateFcn(hObject, eventdata, handles)
% hObject    handle to phaseShift (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in calculate.
function calculate_Callback(hObject, eventdata, handles)
% hObject    handle to calculate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Frequency
f=str2double(get(handles.freq,'string'))*(10^9);
%Element spacing
d=str2double(get(handles.elemSpacing,'string'));
%Scan angle
scan_ang=str2double(get(handles.scanAngle,'string'));

%Processing errors
if (isnan(f))||(isnan(d)||isnan(scan_ang))
    errordlg('All values must be a number','ERROR')
elseif (scan_ang<-90)||(scan_ang>90)
    errordlg('The scan angle must be within the interval [-90,90]','ERROR')
else
    %Speed of light
    c=3*10^8;
    %Wavelength
    lambda=c/f;
    %Number of array elements
    N=8;
    %Calculate the phase shift
    phase_shift=calculate_phaseShift(lambda,d,scan_ang);
    %Print the value into the box
    set(handles.phaseShift,'string',num2str(phase_shift));
    %Calculate the phases of each array element
    phases=calculate_phases(N,phase_shift);
    %If the scan angle is negative, the phases change their order
    if(scan_ang<0)
        phases=toggle(phases);
    end
    %Print the values of all the phases
```

```
    set(handles.phase1,'string',num2str(phases(1)));
    set(handles.phase2,'string',num2str(phases(2)));
    set(handles.phase3,'string',num2str(phases(3)));
    set(handles.phase4,'string',num2str(phases(4)));
    set(handles.phase5,'string',num2str(phases(5)));
    set(handles.phase6,'string',num2str(phases(6)));
    set(handles.phase7,'string',num2str(phases(7)));
    set(handles.phase8,'string',num2str(phases(8)));
    %Calculate the control voltage values needed
    warning('off','MATLAB:dispatcher:InexactCaseMatch');
    %Calculate the corresponding control voltages to each phase
    for i=1:length(phases)
        voltages(i)=get_voltage(phases(i));
    end
    %Print the values of all the voltages sent
    set(handles.voltage1,'string',num2str(voltages(1)));
    set(handles.voltage2,'string',num2str(voltages(2)));
    set(handles.voltage3,'string',num2str(voltages(3)));
    set(handles.voltage4,'string',num2str(voltages(4)));
    set(handles.voltage5,'string',num2str(voltages(5)));
    set(handles.voltage6,'string',num2str(voltages(6)));
    set(handles.voltage7,'string',num2str(voltages(7)));
    set(handles.voltage8,'string',num2str(voltages(8)));
end

% --- Executes on button press in send.
function send_Callback(hObject, eventdata, handles)
% hObject    handle to send (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global serialPort;
% Obtain the voltages should be sent
voltages(1)=str2double(get(handles.voltage1,'string'));
voltages(2)=str2double(get(handles.voltage2,'string'));
voltages(3)=str2double(get(handles.voltage3,'string'));
voltages(4)=str2double(get(handles.voltage4,'string'));
voltages(5)=str2double(get(handles.voltage5,'string'));
voltages(6)=str2double(get(handles.voltage6,'string'));
voltages(7)=str2double(get(handles.voltage7,'string'));
voltages(8)=str2double(get(handles.voltage8,'string'));

% Processing errors
non_valid=0;
incomplete=0;
cont=1;
while (cont<9 && non_valid==0 && incomplete==0)
    if voltages(cont)<0 || voltages(cont)>10
        non_valid=1;
    elseif isnan(voltages(cont))
        incomplete=1;
    end
    cont=cont+1;
end

% Only send the values to the Arduino if the voltage values are from 0 to 10
volts and all the array elements have a voltage assigned
if non_valid==1
    errordlg('Voltages from 0 to 10V','ERROR')
elseif incomplete==1
    errordlg('All the control voltage values must be a number','ERROR')
else
```

```
    % All the voltage values are mapped
    for i=1:length(voltages)
        voltages_mapped(i)=map(voltages(i),0,10,0,255);
    end
    % The bytes representing the voltage values are sent to the Arduino
    portNum=2;
    for i=1:length(voltages_mapped)
        serialPort.analogWrite(portNum,voltages_mapped(i));
        portNum=portNum+1;
    end
    % Calculate the voltage values expected at the analog ports of the micro-
controller
    for i=1:length(voltages_mapped)
        voltages_set(i)=map2(voltages_mapped(i),0,255,0,10);
    end

    % Print these values in the user interface
    set(handles.volt_set1,'string',num2str(voltages_set(1)));
    set(handles.volt_set2,'string',num2str(voltages_set(2)));
    set(handles.volt_set3,'string',num2str(voltages_set(3)));
    set(handles.volt_set4,'string',num2str(voltages_set(4)));
    set(handles.volt_set5,'string',num2str(voltages_set(5)));
    set(handles.volt_set6,'string',num2str(voltages_set(6)));
    set(handles.volt_set7,'string',num2str(voltages_set(7)));
    set(handles.volt_set8,'string',num2str(voltages_set(8)));
end

% --- Executes on button press in info.
function info_Callback(hObject, eventdata, handles)
% hObject    handle to info (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
warndlg('Connect the Arduino with port COM4 before sending values','WARNING');

% --- Executes on button press in get_phases.
function get_phases_Callback(hObject, eventdata, handles)
% hObject    handle to get_phases (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Frequency
f=str2double(get(handles.freq,'string'))*10^9;
% Element spacing
d=str2double(get(handles.elemSpacing,'string'));
% Get the phase shift from the box
phase_shift=str2double(get(handles.phaseShift,'string'));
% Get the phase shift from the box
scan_ang=str2double(get(handles.scanAngle,'string'));
%Number of array elements
N=8;
% Speed of light
c=3*10^8;
% Wavelength
lambda=c/f;
% Error: Not a number
if isnan(phase_shift)
    errordlg('Phase shift must be a number','ERROR')
else
    % Calculate the phase of each array element
    phases=calculate_phases(N,phase_shift);
    % If there is no scan angle introduced
```

```
    if (isnan(f)||isnan(d))
        warndlg('The scan angle cannot be determined.','WARNING')
        set(handles.scanAngle,'string','');
    else
        % Calculates the scan angle
        scan_ang2=asind((phase_shift*lambda)/(360*d));
        if((isnan(scan_ang))||((abs(scan_ang2)~=scan_ang)))
            set(handles.scanAngle,'string',num2str(scan_ang2));
            % If the scan angle is negative change the order of phases
        elseif((abs(scan_ang2)==scan_ang)&& scan_ang<0)
            set(handles.scanAngle,'string',num2str(scan_ang));
            phases=toggle(phases);
        end
    end
end
% Print the values of all the phases
set(handles.phase1,'string',num2str(phases(1)));
set(handles.phase2,'string',num2str(phases(2)));
set(handles.phase3,'string',num2str(phases(3)));
set(handles.phase4,'string',num2str(phases(4)));
set(handles.phase5,'string',num2str(phases(5)));
set(handles.phase6,'string',num2str(phases(6)));
set(handles.phase7,'string',num2str(phases(7)));
set(handles.phase8,'string',num2str(phases(8)));

% --- Executes on button press in get_voltages.
function get_voltages_Callback(hObject, eventdata, handles)
% hObject    handle to get_voltages (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
phases(1)=str2double(get(handles.phase1,'string'));
phases(2)=str2double(get(handles.phase2,'string'));
phases(3)=str2double(get(handles.phase3,'string'));
phases(4)=str2double(get(handles.phase4,'string'));
phases(5)=str2double(get(handles.phase5,'string'));
phases(6)=str2double(get(handles.phase6,'string'));
phases(7)=str2double(get(handles.phase7,'string'));
phases(8)=str2double(get(handles.phase8,'string'));

% Calculate the corresponding control voltages to each phase
warning('off','MATLAB:dispatcher:InexactCaseMatch');

for i=1:length(phases)
    voltages(i)=get_voltage(phases(i));
end

% Print the values of all the voltages sent
set(handles.voltage1,'string',num2str(voltages(1)));
set(handles.voltage2,'string',num2str(voltages(2)));
set(handles.voltage3,'string',num2str(voltages(3)));
set(handles.voltage4,'string',num2str(voltages(4)));
set(handles.voltage5,'string',num2str(voltages(5)));
set(handles.voltage6,'string',num2str(voltages(6)));
set(handles.voltage7,'string',num2str(voltages(7)));
set(handles.voltage8,'string',num2str(voltages(8)));

% --- Executes on button press in reset.
function reset_Callback(hObject, eventdata, handles)
% hObject    handle to reset (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
%Reset phase shift
set(handles.phaseShift,'string','');

% Reset phases values
set(handles.phase1,'string','');
set(handles.phase2,'string','');
set(handles.phase3,'string','');
set(handles.phase4,'string','');
set(handles.phase5,'string','');
set(handles.phase6,'string','');
set(handles.phase7,'string','');
set(handles.phase8,'string','');

% Reset control voltage values
set(handles.voltage1,'string','');
set(handles.voltage2,'string','');
set(handles.voltage3,'string','');
set(handles.voltage4,'string','');
set(handles.voltage5,'string','');
set(handles.voltage6,'string','');
set(handles.voltage7,'string','');
set(handles.voltage8,'string','');

% Reset control voltage values sent
set(handles.volt_set1,'string','');
set(handles.volt_set2,'string','');
set(handles.volt_set3,'string','');
set(handles.volt_set4,'string','');
set(handles.volt_set5,'string','');
set(handles.volt_set6,'string','');
set(handles.volt_set7,'string','');
set(handles.volt_set8,'string','');

function phase8_Callback(hObject, eventdata, handles)
% hObject    handle to phase8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of phase8 as text
%        str2double(get(hObject,'String')) returns contents of phase8 as a double


% --- Executes during object creation, after setting all properties.
function phase8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to phase8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function phase7_Callback(hObject, eventdata, handles)
% hObject    handle to phase7 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of phase7 as text
%        str2double(get(hObject,'String')) returns contents of phase7 as a double


% --- Executes during object creation, after setting all properties.
function phase7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to phase7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function phase6_Callback(hObject, eventdata, handles)
% hObject    handle to phase6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of phase6 as text
%        str2double(get(hObject,'String')) returns contents of phase6 as a double


% --- Executes during object creation, after setting all properties.
function phase6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to phase6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function phase5_Callback(hObject, eventdata, handles)
% hObject    handle to phase5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of phase5 as text
%        str2double(get(hObject,'String')) returns contents of phase5 as a double


% --- Executes during object creation, after setting all properties.
function phase5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to phase5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function phase4_Callback(hObject, eventdata, handles)
% hObject    handle to phase4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of phase4 as text
%        str2double(get(hObject,'String')) returns contents of phase4 as a double


% --- Executes during object creation, after setting all properties.
function phase4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to phase4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function phase3_Callback(hObject, eventdata, handles)
% hObject    handle to phase3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of phase3 as text
%        str2double(get(hObject,'String')) returns contents of phase3 as a double


% --- Executes during object creation, after setting all properties.
function phase3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to phase3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function phase2_Callback(hObject, eventdata, handles)
% hObject    handle to phase2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of phase2 as text
%        str2double(get(hObject,'String')) returns contents of phase2 as a double


% --- Executes during object creation, after setting all properties.
function phase2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to phase2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function phase1_Callback(hObject, eventdata, handles)
% hObject    handle to phase1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of phase1 as text
%        str2double(get(hObject,'String')) returns contents of phase1 as a double


% --- Executes during object creation, after setting all properties.
function phase1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to phase1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function voltage8_Callback(hObject, eventdata, handles)
% hObject    handle to voltage8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of voltage8 as text
%        str2double(get(hObject,'String')) returns contents of voltage8 as a
double


% --- Executes during object creation, after setting all properties.
function voltage8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to voltage8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function voltage7_Callback(hObject, eventdata, handles)
% hObject    handle to voltage7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of voltage7 as text
%        str2double(get(hObject,'String')) returns contents of voltage7 as a
double


% --- Executes during object creation, after setting all properties.
function voltage7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to voltage7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function voltage6_Callback(hObject, eventdata, handles)
% hObject    handle to voltage6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of voltage6 as text
%        str2double(get(hObject,'String')) returns contents of voltage6 as a
double


% --- Executes during object creation, after setting all properties.
function voltage6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to voltage6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function voltage5_Callback(hObject, eventdata, handles)
% hObject    handle to voltage5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of voltage5 as text
%        str2double(get(hObject,'String')) returns contents of voltage5 as a
double


% --- Executes during object creation, after setting all properties.
function voltage5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to voltage5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function voltage4_Callback(hObject, eventdata, handles)
% hObject    handle to voltage4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of voltage4 as text
%        str2double(get(hObject,'String')) returns contents of voltage4 as a
double


% --- Executes during object creation, after setting all properties.
function voltage4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to voltage4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function voltage3_Callback(hObject, eventdata, handles)
% hObject    handle to voltage3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of voltage3 as text
%        str2double(get(hObject,'String')) returns contents of voltage3 as a
double


% --- Executes during object creation, after setting all properties.
function voltage3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to voltage3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function voltage2_Callback(hObject, eventdata, handles)
% hObject    handle to voltage2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of voltage2 as text
%        str2double(get(hObject,'String')) returns contents of voltage2 as a
double


% --- Executes during object creation, after setting all properties.
function voltage2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to voltage2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function voltage1_Callback(hObject, eventdata, handles)
% hObject    handle to voltage1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of voltage1 as text
%        str2double(get(hObject,'String')) returns contents of voltage1 as a
double


% --- Executes during object creation, after setting all properties.
function voltage1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to voltage1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function volt_set8_Callback(hObject, eventdata, handles)
% hObject    handle to volt_set8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of volt_set8 as text
%        str2double(get(hObject,'String')) returns contents of volt_set8 as a
double


% --- Executes during object creation, after setting all properties.
function volt_set8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to volt_set8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function volt_set7_Callback(hObject, eventdata, handles)
% hObject    handle to volt_set7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of volt_set7 as text
%        str2double(get(hObject,'String')) returns contents of volt_set7 as a
double


% --- Executes during object creation, after setting all properties.
function volt_set7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to volt_set7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function volt_set6_Callback(hObject, eventdata, handles)
% hObject    handle to volt_set6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of volt_set6 as text
%        str2double(get(hObject,'String')) returns contents of volt_set6 as a
double
```

```
% --- Executes during object creation, after setting all properties.
function volt_set6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to volt_set6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function volt_set5_Callback(hObject, eventdata, handles)
% hObject    handle to volt_set5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of volt_set5 as text
%        str2double(get(hObject,'String')) returns contents of volt_set5 as a
double


% --- Executes during object creation, after setting all properties.
function volt_set5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to volt_set5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function volt_set4_Callback(hObject, eventdata, handles)
% hObject    handle to volt_set4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of volt_set4 as text
%        str2double(get(hObject,'String')) returns contents of volt_set4 as a
double



% --- Executes during object creation, after setting all properties.
function volt_set4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to volt_set4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end


function volt_set3_Callback(hObject, eventdata, handles)
% hObject    handle to volt_set3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of volt_set3 as text
%        str2double(get(hObject,'String')) returns contents of volt_set3 as a
double


% --- Executes during object creation, after setting all properties.
function volt_set3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to volt_set3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function volt_set2_Callback(hObject, eventdata, handles)
% hObject    handle to volt_set2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of volt_set2 as text
%        str2double(get(hObject,'String')) returns contents of volt_set2 as a
double


% --- Executes during object creation, after setting all properties.
function volt_set2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to volt_set2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function volt_set1_Callback(hObject, eventdata, handles)
% hObject    handle to volt_set1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of volt_set1 as text
```

```
%        str2double(get(hObject,'String')) returns contents of volt_set1 as a
double


% --- Executes during object creation, after setting all properties.
function volt_set1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to volt_set1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
clear serialPort
delete(instrfind({'Port'},{'COM4'}));
% Hint: delete(hObject) closes the figure
delete(hObject);
```

Appendix D : Design Wilkinson details using AWR

In chapter 5 the parallel feed network of a phased antenna was designed. This network was composed by a network of 3dB/0º Wilkinson power dividers connected forming a tree. The schematic of a real 3dB/0º Wilkinson is shown in the figure D.1. In order to calculate the physical length and the width of each track, the tool TXLine is used. After that the tune tool is also used to optimize the frequency response as desired. The design of the power dividers at all the stages will have the same schematic, but some transmission lines will be different.

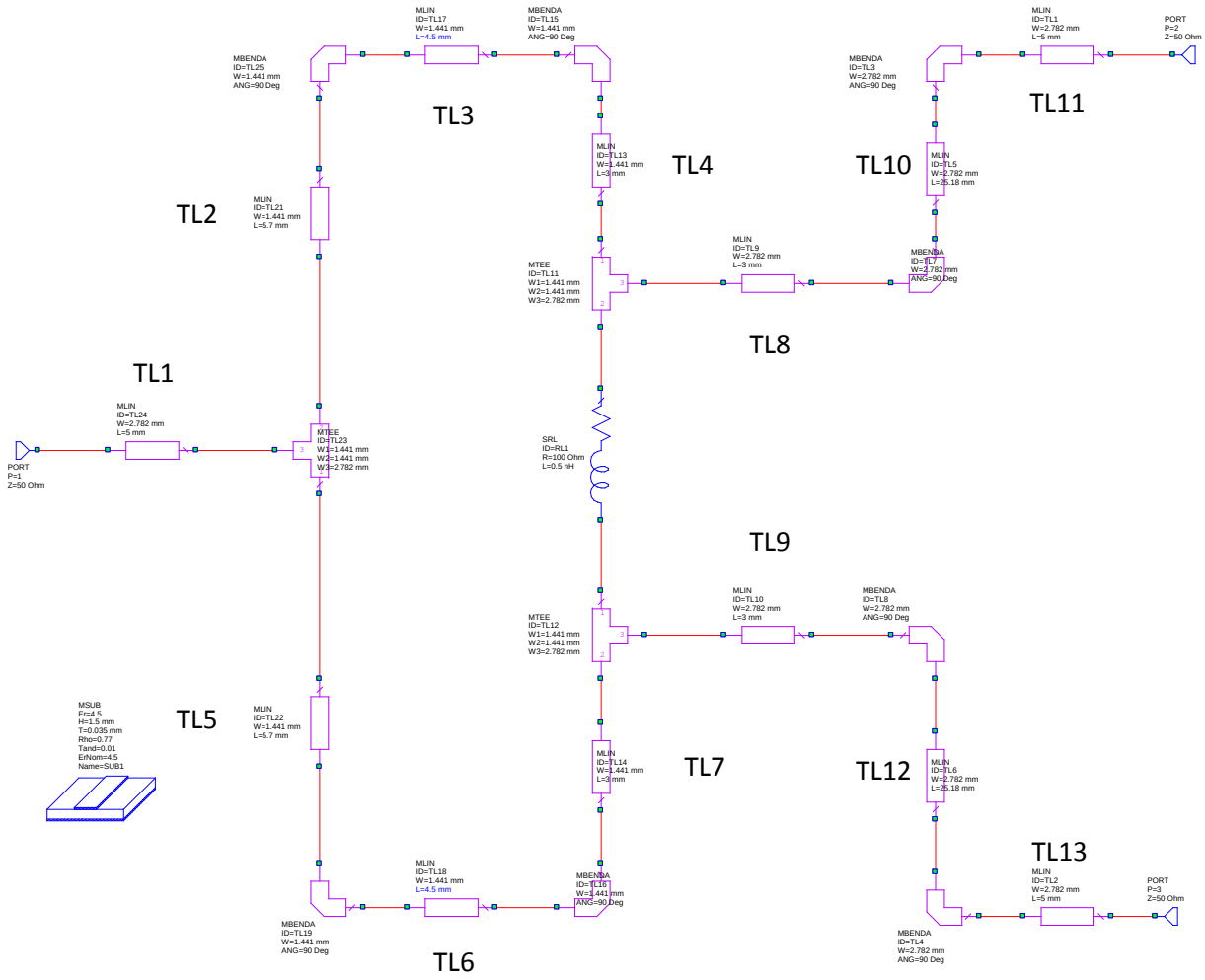


Figure D.1. Schematic of a real microstrip 3dB/0º Wilkinson.

Using the TXLine tool the physical length and the width can be calculated. The input parameters are the characteristics of the substrate used, electric length, frequency and characteristic impedance of the transmission line.

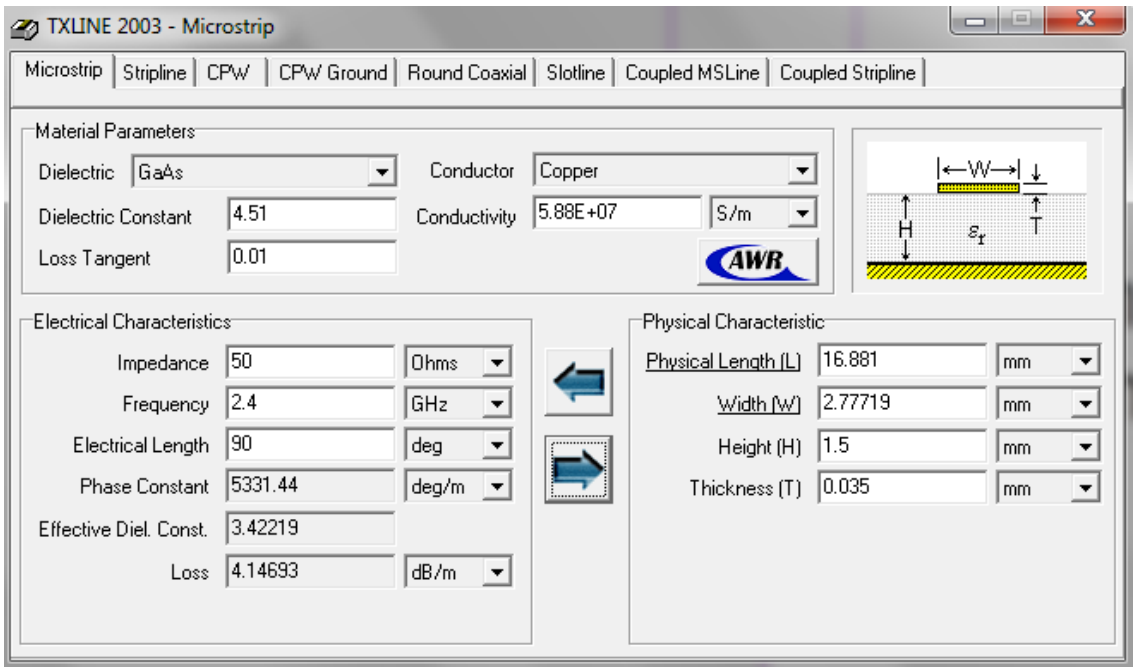


Figure D.2. Sample of the TXLine tool calculation.

As mentioned in the figure 5.2 two types of transmission lines with different characteristic impedance are used to implement the power divider. The characteristic impedance could be obtained varying the width of the transmission lines. Using TXLine the values obtained are the following:

Characteristic Impedance	Width
$Z_{o1} = 50\ \Omega$	2.782 mm
$Z_{o2} = \sqrt{2}Z_{o1}$	1.441 mm

Table D.1. Calculation of transmission line width using TXLine.

D.1. 3dB/0º Wilkinson at first stage

As depicted in the figure 5.2, there are two transmission lines with an electric length of 90 degrees. Using TXLine the physical length is calculated and divided into different transmission lines. These two 90º transmission lines are composed by TL2, TL3, TL4 and TL5, TL6, TL7 respectively. The input and output ports have a characteristic impedance of 50Ω. The physical

length of the transmission lines composing these ports does not vary the frequency response. Hence, the physical length of the input port composed by the transmission line TL1 will be short in order to minimize the losses. Depending on the stage where the power divider is placed at, the output ports should have different lengths. At the first stage, considering that the element spacing of the array is $\lambda/2$ and considering the layout shown in the figure 5.10, the lines TL10 and TL12 should have a determined length. After some calculations the values of the physical lengths are obtained. All the physical length and width of the transmission lines composing the power divider are noted in the following table.

Symbol	Physical Length	Width
TL1	5 mm	2.782 mm
TL2	5.7 mm	1.441 mm
TL3	4.5 mm	1.441 mm
TL4	3 mm	1.441 mm
TL5	5.7 mm	1.441 mm
TL6	4.5 mm	1.441 mm
TL7	3 mm	1.441 mm
TL8	3 mm	2.782 mm
TL9	3 mm	2.782 mm
TL10	25.18 mm	2.782 mm
TL11	5 mm	2.782 mm
TL12	25.18 mm	2.782 mm
TL13	5 mm	2.782 mm

Table D.2. Physical length and width of lines composing the real Wilkinson at the first stage.

D.2. 3dB/0º Wilkinson at second stage

The overall values of the physical length and width of the transmission lines are the same as those from the Wilkinson at the first stage, except the transmission lines forming the output ports. As depicted in the figure 5.12, the length of the transmission lines TL10 and TL12 corresponds to the values shown in the following table. These lengths have been calculated to set the element spacing of the first stage to $\lambda/2$.

Symbol	Physical Length	Width
TL10	56.43 mm	2.782 mm
TL12	56.43 mm	2.782 mm

Table D.3. Physical length and width of lines composing the real Wilkinson at the second stage.

D.3. 3dB/0º Wilkinson at third stage

The third stage is composed by only one power divider. The transmission lines composing the output ports will have the values shown in the following table.

Symbol	Physical Length	Width
TL10	118.9 mm	2.782 mm
TL12	118.9 mm	2.782 mm

Table D.4. Physical length and width of lines composing the real Wilkinson at the second stage.

All the other values of the physical length and width of the transmission lines are the same as those from the Wilkinson at the first or second stage.

BIBLIOGRAPHY

[1] Merril I. Skolnik, “Radar Handbook”, Third edition 2008, Mc. Graw Hill, chapters 1 and 13.

[2] Bassem R. Mahafza, “Radar Systems Analysis and Design Using MATLAB” 2000, Chapman & Hall/CRC, section 10.4.

[3] Jin Au Kong, “Electromagnetic Wave Theory”, Second edition 2008, pp. 516-541.

[4] Constantine A. Balanis, “Antenna Theory”, Second edition 1997, John Willey & Sons, pp. 28-45

[5] Warren L. Stutzman and Gary A. Thiele, “Antenna Theory and Design”, Third edition, John Willey & Sons, pp. 23-60, pp. 122-123, pp. 272-282, pp. 320-325.

[6] David M. Pozar, “Microwave Engineering”, Fourth edition 2011, John Willey & Sons, pp. 328-333.

[7] Danial Ehyaie, “Novel Approaches to the Design of Phased Array Antennas”, Electrical Engineering Dissertation, University of Michigan, 2011, pp. 10-13.

[8] Amandeep Kaur, “Ellectronically Steerable planer Phased Array Antenna”, International Journal of Engineering Trends and Technologie, Volume 3, Issue 6 2012, pp.708-709.

[9] E. Brookner, "Phased-array radars," Scientific American, vol. 252, 1985.

[10] Allan R. Hambley, “Electronics”, Second edition, Prentice Hall, 1999, pp. 64-108.

[11] Electronic and Control Systems Department, “PSpice Notes Version 9.1”, Technical College of Madrid.

[12] Vicente Gonzalez Posadas, “Radio-communication Technology Practices”, Audiovisual and Communication Department, 2002, pp. 25-31.

[13] R. J. Mailloux and I. Books, Phased array antenna handbook: Artech House, 2005.

[14] MATLAB Tutorials and Learning Resources, MathWorks. Available: http://www.mathworks.com/academia/student_center/tutorials/launchpad.html

[15] Graph_picker MATLAB Function, MathWorks. Available: http://www.mathworks.com/matlabcentral/fileexchange/41313-graph-picker

[16] MATLAB Support Package for Arduino. Available: http://www.mathworks.com/matlabcentral/fileexchange/32374-matlab-support-package-for-arduino-aka-arduinoio-package

[17] Phase Shifter HMC928LP5E Datasheet. Available: http://www.hittite.com/content/documents/data_sheet/hmc928lp5.pdf

[18] Arduino Mega 2560 Schematic. Available: http://arduino.cc/en/uploads/Main/arduino-mega2560-schematic.pdf

[19] Arduino Mega 2560 Board Characteristics. Available: http://arduino.cc/en/Main/arduinoBoardMega2560

[20] Brian W. Evans, "Arduino Programming Notebook", Second Edition 2008.

[21] MATLAB Summary and Tutorial. Available: http://www.math.ufl.edu/help/matlab-tutorial/

[22] Operational Amplifier μA741C .Available: http://www.ti.com/lit/ds/symlink/ua741.pdf